UNIVERSITY OF OSLO
Department of Informatics

Bayllocator: A proactive system to predict server utilisation and dynamically allocate memory resources using Bayesian networks and ballooning

Evangelos Tasoulas
evanget@ifi.uio.no

Network and System Administration

Oslo University College

May 23, 2012

Bayllocator: A proactive system to predict server utilisation and dynamically allocate memory resources using Bayesian networks and ballooning

Evangelos Tasoulas
evanget@ifi.uio.no

Network and System Administration
Oslo University College

May 23, 2012

Abstract

With the blossom of virtualization and cloud computing, virtualized systems can be found from small companies to service providers and big data centers. All of them use this technology because of the many benefits it has to offer, such as a greener ICT, cost reduction, improved profitability, uptime, flexibility in management, maintenance, disaster recovery, provisioning and more. The main reason for all of these benefits is server consolidation which can be improved with dynamic resource allocation techniques. Out of the resources to be allocated, memory is one of the hardest and it needs proper planning and ideally good predictions and proactivity. Many attempts have been made to approach this problem, but most of them are using traditional statistical mathematical methods. In this thesis the application of discrete Bayesian networks is evaluated, to offer probabilistic predictions on system utilisation with focus in memory. Bayllocator is built to provide proactive dynamic memory allocation based on the Bayesian predictions, for a set of virtual machines running in a single hypervisor. The results show that with proper tuning Bayesian networks are capable of providing good predictions for system load, and increase performance and consolidation of a single hypervisor. Their modularity gives great freedom for experimentation, and with small changes in nodes and the causal relationship of the nodes, different results can be obtained.

Acknowledgements

It is an honour for me to express my appreciation to the following people and recognise their support in many different ways:

- First of all I would like to thank my beloved family including mother, father, sister and uncle George, because I would not be able to be who I am and at the place I am right now without their total support (emotional and economic).
- Norway, University of Oslo, and Oslo and Akershus University College which provided a top quality study environment and facilities.
- Hårek Haugerund who is a great person, professor, supervisor and spent a significant amount of his time for me and this thesis. He is always the one giving hope to the students with his smile even when things are not as they are supposed to be.
- Ismail Hassan for being there for me always when I needed him with rich discussions on different subjects and also for this thesis. Moreover, for supporting me as a student assistant at two of his courses.
- Kyrre Begnum who is one of the most knowledgeable and exciting professors I have ever met, and he helped with the submission of a paper created from this thesis in LISA '12 conference. He also provided some precious comments and corrections.
- Æleen Frisch for providing me with feedback for this thesis, and for being a supportive and very approachable person from the very first moment we met at LISA Conference '11.
- Dr Grigorios Koulouras for his friendship, trust, and help in problem solving always when I need it.
- Prof Constantinos Nomicos and Mr George Stamatoukos for their trust in my potential, when I could not even trust myself back in the Bachelor studies.
- Marco Scutari, author of bnlearn R package and Søren Højsgaard, author of gRain R package for quickly answering a couple of questions by email, helping me to advance quickly when I was stuck.
- Maria Roumpoutsou, Filippia Zikou and Nikos Roumpoutsos for their math related help when needed especially in the previous semester.
- Inkscape and RStudio for making my life easier throughout this project.
- Dimitris Agiakatsikas and Anastasios Veresses for being two best friends and still keeping in touch even if the distance is long.
- Anyone else who read this thesis and gave me some precious feedback.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the introduction chapter a motivation section is explaining why the dynamic memory allocation using Bayesian networks have been chosen in this project. The problem statement with some research questions is defined and the structure of the rest of this thesis is briefly described.

1.1 Motivation

A Statistical research (2007) provided that we would have 1 billion personal computers by 2008 [1]. This was confirmed by another research in June, 2008 and both of them foresees that this number will be doubled by 2014-2015 [2].

The adoption from the Enterprise increases as well as ICT increases corporate productivity, but the companies often need to use non stop running servers [3]. This leads to increased hardware purchase costs, as well as peripheral costs and environmental damage coming from the power consumption of the servers and the cooling facilities [4]. If one considers that most of the time (>70%) in typical deployments servers are idle, the cost to operate underutilised servers is significant [4]. Furthermore, if a server requires more computing power at peak load and not for more than a few hours every day, either better hardware has to be acquired, or more servers in a load balanced topology have to be added. As an outcome, even more power consumption and idle time is added to the infrastructure. Except that, human presence and intervention are needed to take care of the hardware changes.

A solution to the described problem can be given using by server consolidation and Virtualization technology. Virtualization brings many benefits such as reduction of power consumption, total cost of operation and maintenance, flexibility and scalability, by breaking the one by one (1:1) relationship between the operating system (OS) and the hardware [5]. In simple words, many operating systems or virtual machines can run on top of a single physical machine called hypervisor. The amount of the running virtual machines is limited by

the physical hardware resources of the hypervisor. Using virtualization results to better resource utilisation and less power consumption, by reducing the number of physical machines and CPU idle time [6].

Traditionally a company would have to buy three servers if they needed one server for backup, one for directory services and one for a private branch exchange (PBX). If they decide to go for virtualization, they would need only one powerful server. This may have a potentially higher initial investment as this server needs to be able to handle the load of the three virtual servers, but it will usually pay off in a short term by reducing the electricity and maintenance expenses [6]. These benefits and the positive impact of virtualization bring this technology not only to the server, but to the desktop market as well. A research shows that more than 70 million desktops will be virtualized by 2014 [7].

Since the virtual machines have no physical boundaries, they maintain great flexibility to manipulation of their consumable resources and extend the options to provide high Quality of Service (QoS) using scripts or other soft technologies. Some techniques to address QoS using virtualization may be:

- The same as in real hardware, by shutting down the virtual machine and add more hardware resources but in a more flexible way as the hardware is virtual. Of course the physical hypervisor has to be able to provide the increased virtual hardware resources or the virtual machine has to be moved to a more powerful hypervisor.
- Live migration between different hypervisors to achieve better resource utilisation [8, 9, 10].
- Use common resources such as memory duplication when this is possible [11].
- Dynamically live resizing of local resources in the virtual machines with hot add/remove memory and CPU scheduling prioritization [12].
- Any possible combination of the above [12]

From the above mentioned QoS solutions, the first one needs manual user intervention and the demand to power off the virtual machines. This might be an option but not very flexible as it is not dynamically adapted to the workload. Automated live migration to a stronger hypervisor is a much better alternative, but it needs further infrastructure like the existence of a common storage device (SAN/NAS) and more than one physical hosts - hypervisors [10]. Small businesses (SMBs) might not even have this additional infrastructure or spare servers to use live migration. The answer then lies on the 3rd and 4th option. Try to get the maximum of the word "virtual", by dynamically live changing the resources allocated to the virtual machines in a single hypervisor timely, to achieve optimal performance.

Dynamic CPU and live Memory scaling are provided by mainstream virtualization technologies like KVM, XEN and VMware [13, 14]. CPU is a time-

shared resource so it can be easily shared among the virtual machines and priority can be given by the scheduler to most important virtual machines [13]. Memory on the other hand is harder to share, since a memory page reserved by one virtual machine cannot be released as long as it is in use. Common tactics used for memory sharing and scaling are achieved by memory overcommitment. Memory overcommitment builds upon the assumption that all virtual machines lying in the same hypervisor do not need to consume 100% of their available memory at all times. Following this assumption, more memory than the total memory available in the hypervisor is assigned to the total number of virtual machines and some memory addresses are used by more than one guest, usually not at the same time.

With techniques like ballooning, memory hotplug and Shared Memory or Samepage Merging the memory will be distributed between virtual machines on demand [15, 14, 11]. Shared memory and Samepage Merging is referred to the same technique and the hypervisor will try to merge redundant memory copies shared by multiple virtual machines. If many virtual machines in the same host running similar kind of software, the samepage merging feature might lead to much improved memory utilisation. Otherwise it will not have a big impact.

In the ballooning technique, a driver is installed in the guest OS and communicates with the hypervisor. When the hypervisor needs more memory, it will ask for it and the balloon driver will inflate in the guest, trying to allocate memory pages from within the virtual machine and give it back to the host. When the host has available memory it will deflate the balloon and the claimed memory will be available again to the guest OS. The guest OS will not notice any memory changes other than that a process (the balloon driver) will need to allocate more of its memory and in the worst case it will need to start swapping [13, 15].

In memory hot plugging, the guest will see its total memory increasing as if it was added a new memory DIMM. In the unplugging, the guest OS will be forced to free some of its memory (and possibly swap it in the hard disk), and it will see its total memory decreasing as if a memory DIMM was removed.

In this thesis focus and greater exploration will be given to the open source Kernel Virtualization Module (KVM) of the Linux kernel and its memory scaling abilities using the ballooning technique. VMware is a paid proprietary product so it was excluded early and XEN was discarded in this exploratory thesis in favour of KVM, as XEN's late admission to mainstream Linux kernel led some major companies and their widely adopted Linux distributions to choose KVM instead [16, 17, 18].

At the time of writing, KVM does not support dynamic memory ballooning while XEN (at least the commercial version) and VMware does [19, 20, 21, 22]. It requires external control and the only publicly available solution found so far is MOM [23]. MOM stands for Memory Overcommitment Manager and it will try to optimize ballooning and KSM (Kernel Samepage Merging) in real

time according to a given policy. This makes MOM a reactive tool to excessive memory load. It is worthwhile to mention that even XEN and VMware that they support dynamic memory overcommitment, their algorithms are also reactive.

One con of the reactive ballooning is that even if the virtual machine is trying to collaborate, the balloon driver might not be able to recover the memory requested by the host fast enough to avoid performance degradation [13, 15]. To avoid or reduce this undesirable behaviour some proactivity is needed. A tool to be able to learn from repeatable events, predict virtual machine memory load beforehand and act. This would help to solve in a large degree this con of real time ballooning.

The predictability of server load is not something new. It is highly feasible by using statistical methods and it has been tried and proved that it works many times in different scenarios in computing [24, 25, 26]. This is sensible as many real life events are recurring, thus highly predictable. Their recurrence and predictability can be transferred and observed in server utilisation as well. For a Norwegian web server serving Norwegian news in the Norwegian language, it is almost sure that it will have higher load during the Norwegian day time. It is also highly probable that it will not have the same high load during holidays or weekends as people tend to go to trips and relax these days. A directory server or a file server of a company, will usually have much higher load during the standard working hours of the weekdays that the majority of the employees is working. A backup server usually gets no holidays. It might be scheduled to take backups every night 365/366 days a year. These are only some examples of predictable computing utilisation.

The scope of this thesis will be an attempt to create a prototype self learning, predictive system for dynamic memory allocation in virtual machines running in a single KVM hypervisor using Bayesian networks. A Bayesian belief network is a probabilistic graphical directed acyclic model, indicating the conditional dependence in a set of random variables [27, 28]. Bayesian networks are popular to probabilistic artificially intelligent decision support systems for their good ability to learn from new observations, perceive and plan ahead [29, 30, 31, 32]. A Bayesian network first needs to be trained by a person who is knowledgeable in the environment it will have to work with. Nodes of random variables (chosen by the trainer) have to be created and their directed relationship influence needs to be defined. The Bayesian network will then provide probabilities for the event represented by a node to happen, given the probabilities of the surrounding nodes. The more knowledge the trainer has, the better the belief network will become eventually as the node relationships will be more tuned. More nodes for more accurate results can be added to the Bayesian network if there is evidence of interference from other sources and the current ones can adapt to behavioural changes of the system giving accurate results as more knowledge is absorbed. This means the longer the system works, more confident results will be provided.

Bayesian networks can be found in different applications varying from weather

forecasting, biomedical research and more in need for decision making and forecasting [33, 34, 35]. In ICT, Bayesian networks have been used for speech recognition, e-mail SPAM filtering, Internet traffic classification, prediction of potential problems to avoid catastrophic failures, adaptive routing provision for mobile ad hoc networks (MANETs) and more [36, 37, 38, 39, 40, 26, 41].

1.2 Problem Statement

To create a proactive probabilistic self learning dynamic resource scaling system, focused on memory overcommitment using ballooning and control the balloons with input from a Bayesian network.

1. How can a Bayesian network be trained to predict server utilisation?
2. What kind of data should the system be fed with?
3. What kind of workload is this system able to predict?
4. How can the system learn from events to improve its results?
5. How to use the output of such a system to dynamically balance memory for virtual machines?

1.3 Thesis Structure

The layout of this thesis is as follows: the background chapter 2 comes after this introduction where related work and literature is collected. A simple Bayesian network example is given and solved analytically to give a better understanding of how the final system will make the predictions. The methodology chapter 3 next gives an explanation of the methods used and some important scripts related to the method and it is followed by the results and analysis chapters (4, 5) where the actual results are displayed and analysed in detail. Then in the discussion chapter 6, an overall evaluation of what has been done, problems encountered and future work is discussed, and last comes the conclusion where the questions asked in the problem statement section 1.2 are answered clearly. Also an appendix chapter with all of the important scripts created can be found at the very end of this document.

Chapter 2

Background and literature

In the background chapter a study and review of related literature and scientific ongoing work for the underlying technologies used in this thesis takes place. This involves virtualization which is the core platform to work on, memory overcommitment techniques and Bayesian inference and networks. A simple practical analytical example on how a Bayesian network could work to predict server utilisation is explained.

2.1 Virtualization

As described in a paper back in 1991, *Virtualization may be defined as the process by which a human viewer interprets a patterned sensory impression to be an extended object in an environment other than that in which it physically exists* [42].

The term Virtualization in computing refers to emulation of hardware, software or services to allow abstraction and isolation of lower level functionalities and underlying hardware or operating systems. Consequently, physical resource sharing, portability, manageability, flexibility, migration and hardware or operating system independence is achieved in the higher level functions [43, 44, 45, 46].

Hardware virtualization (or sometimes referred to it as server virtualization), is by far the most common application of virtualization technology today. Usually when people use the term ”virtualization” they mostly refer to hardware virtualization [47].

2.1.1 Beyond the Bounds of Server Consolidation

Server consolidation is the main field hardware virtualization was used since its early appearance back in 1960, and still this is the dominant selling market [5, 47]. The technology has matured enough lately to pass to the end

desktop users and lower end server environments used even in SMBs [48]. The benefits of hardware virtualization are many. The ability to break the one by one relationship between applications and OS (operating systems) and between the OS and the hardware is the one at a glance [5]. This is achieved by physical resource sharing and strong isolation which also leads to strong security [5, 43]. An expected effect of the one by one breakage is also lower power consumption, both from the servers themselves and the facility cooling systems [4, 47].

2.1.2 Turning Green

A big challenge for many organisations today is to find affordable solutions to help them operate "green" by reducing their CO_2 emissions. Most of those emissions are directly implicated with energy consumption [49]. Green Computing is the term used in the computing world to make ICT infrastructure more environmentally friendly by reducing the emitted carbon dioxide. Since power consumption is the main cause of it, virtualization is one of the ICT technologies closely related to this term as physical hardware can be dramatically reduced, hereby, power consumption too [50, 49].

2.1.3 Portability - Migration

Since the OS or applications executed on these virtual machines are separated from the underlying hardware resources, portability, flexibility, migration and manageability are some more benefits instantly coming into the foreground. The virtual hardware of a virtual machine can run seamlessly in different physical hardware behaving exactly in the same way, making the virtual machines easily portable. This is not easy to happen with physical hardware as the target hardware has to have exactly the same specifications to avoid any software breakage. Easy portability leads to flexibility and gives more options when the need comes to upgrade the physical resources or infrastructure. In case of maintenance, migration of the virtual machines into other physical servers can be achieved with a small downtime, or even live migration if there is a shared storage, leading to downtimes as low as 60ms. This is considered almost as zero downtime [10].

2.1.4 Dynamic Resource Allocation and Placement

Most of the time in typical server deployments the servers are underutilised or idle. This leads to increased energy consumption and therefore costs, as low utilisation translates into more physical machines. More physical machines needs more labour, cooling systems that they also consume power and more floor space to be stored increasing expenses and pollution even more [8].

With the use of virtualization, advantage of the low utilisation per server can be taken to reduce the number of physical servers. In an example with 4 physical servers running 6 virtual machines, those can be live migrated around the physical hardware depending their utilisation demand. Moreover, dynamic resource allocation can be used to increase or decrease the virtual resources of each of the virtual machines, given its current load while fulfilling the QoS (Quality of Service) policy [8, 12].

In the mentioned example with the 4 physical servers and 6 virtual machines, a realistic scenario would be to have 3 virtual machines running in the 1st real server, 2 of them running in the 2nd real server and the last 1 being the most demanding of them, running on the 3rd real server on its own. The 4th real server could be shut off or idle (might be a backup). In case that the VM running alone in the 3rd server reduces its working load, it could be transferred in the 2nd server so that the 3rd server could be idle or shut off too. On the other hand, if the load is on its peak, one of the three VMs running in the 1st server could be transferred in the 4th server.

The result is 6 servers running on top of 4 physical servers providing also high availability features. Even if one of the real servers goes down due to hardware issues, the virtual servers will keep on running on the rest of the 3 real servers still in operation. For the same scenario without the use of Virtual Machines, one would need at least 6 real servers without the high availability feature available.

This is the context where this thesis fits in. Try to create a predictive system for dynamic resource allocation for the KVM hypervisor. Most of the available solutions found use real time monitoring and this can lead to performance penalties if the guest systems cannot cooperate to change their allocated resources fast enough.

2.1.5 Backup and Snapshots

In computing storage, snapshot is the frozen state of a system at a distinct point in time [51]. Virtual machine hypervisors have the ability to keep trees of snapshots of their virtual machine guests giving to the user the opportunity to go back at any time. Think of it as if one would keep a ghost image of a hard disk drive. Making changes to the OS and then reverting back to the ghost image. With real hardware this is not a simple procedure and one would need many different hard drives to keep different tree states of the system. This is a very handy feature of the virtual machines that it can be used either for security reasons (got compromised? go back in a clear state and patch the system), or to improve the test and development speed of software or other solutions. There is no need for OS re-installation each time something new is being tested.

Similar to snapshots, the backup functionality becomes trivial when using virtual machines. Instead of keeping single file configuration backups, the whole

virtual system can be backed up (either by taking a copy or a snapshot) so both the backup and disaster recovery procedures get simplified.

2.1.6 Desktop Virtualization

Desktop (or client) virtualization separates the desktop environment of a personal computer from a physical machine using a client-server model. It is a technique similar to the traditional thin clients, combining all of the advantages of thin client functionality and virtualization can offer. Typical implementations of desktop virtualization store the resulting virtualized desktop computer on a remote central server. That means that the user is in front of dummy terminal with a monitor, keyboard, mouse and some peripherals. Old computers or laptops can be re-used for this role resulting in cost reduction by only needing to invest on server infrastructure. As every Desktop is stored in the server, central management and security policies can be applied resulting in a further maintenance cost reduction, and the users can access their desktop virtually from everywhere [52]. These are quite obvious benefits and this is why virtualization is emerging to the desktop market as well and more than 70 million desktops are expected to be virtualized by 2014 [7].

2.1.7 Security by Isolation

Security by isolation is one more of the highlights virtualization can offer. This is achieved by strong isolation of resources between the virtual machines [5, 43] where different services are running. A good example of creativity and how virtualization can be used to achieve better security is "Qubes OS" [53]. Qubes OS is an operating system which isolates applications by running them inside different virtual machines. So in the unpleasant case that the operating system got compromised because of a security bug in a software, the attacker will be locked to the virtual container this software runs in and consequently the threat is vastly reduced.

2.2 Memory Overcommitment

Memory overcommitment as derived from the name, refers to techniques for over subscribing available memory. The philosophy rests on the conjecture that all virtual machines lying on top of one hypervisor, do not need to consume all of their available memory at all times except when they run at peak load. As an instant effect of this assumption, more than the total memory available in the hypervisor is assigned to the total number of virtual machines and when the memory of one guest is not in use, it will be used by other virtual machines on demand. As an example, if a physical hypervisor has 4GB of memory and runs 4 virtual machines, 1.5GB could be given to them or any other combination resulting in a total of 6GB or more.

There are 4 well known tactics to achieve memory overcommitment in virtualization.

1. Swapping
2. Page Sharing or Memory Sharing
3. Ballooning
4. Memory hotplug

Ballooning and Memory sharing are the most frequently used modern memory overcommitment techniques in virtualization.

2.2.1 Swapping

Swapping is the traditional way to support memory overcommitment for many years, not only for virtual machines but also for applications running in the operating system. When the hypervisor is under memory pressure it will choose some memory pages (usually the Least Recently Used - LRU ones) from the guests and write them to hard disk, releasing the memory to be used. When a guest requires memory that has been swapped, the hypervisor reads this memory pages from the hard disk.

Swapping is always tried to be avoided, as it will devastate the performance of the virtual machines and even worse the hypervisor itself, since the hard disk is several orders of magnitude slower than the memory chips. The memory access time is measured in nanoseconds while the hard disk access time is measured in milliseconds.

2.2.2 Shared Memory - Page Sharing

Shared memory, Page sharing or Samepage Merging all refer to the same technique for sharing memory between virtual machines running in the same hypervisor. Essentially, the hypervisor will try to merge identical memory copies shared by multiple virtual machines. If many guests in the same hypervisor run similar kind of operating system or software, the samepage merging feature can improve memory density a lot. KVM uses the KSM module of the Linux kernel to achieve samepage merging as the KVM virtual machines work like single system processes [11]. There is no need for additions or any guest OS modification to utilise samepage merging.

2.2.3 Memory Hotplug

Memory hot plug is a feature supported in very expensive physical hardware, but in virtualization it can work for any virtual machine if memory hot plug is supported by the kernel of the guest operating system. The essence of memory

hot plug add is to add a memory chip (DIMM) in the system so that the operating system will notice a total memory increase and hot plug remove is the same procedure vice versa so the guest OS will notice a total memory decrease as if a memory chip was removed [15].

2.2.4 Ballooning

Ballooning is a technique introduced for the first time in VMware ESX product back in 2002 [14]. Then XEN hypervisor and much later KVM introduced this feature as well [54, 55]. Ballooning needs a driver to be installed in the guest operating system. The driver communicates directly with the hypervisor and acts on demand.

When the hypervisor needs more memory for a guest, it will choose a virtual machine which is the "victim" or the "donor" and the balloon driver will "inflate" in this virtual machine, trying to allocate memory pages from it and give them back to the hypervisor. Then the hypervisor will "deflate" the balloon of the guest who is the "beneficiary" and needs to get the claimed memory, making available these memory addresses to it (see figure 2.1). Both operating systems of the donor and beneficiary guests will not notice any memory changes in their total memory available, other than that a process (the balloon driver) allocates more of its memory [13, 15]. If the balloon driver is very demanding, the guest whose the memory was claimed from will start swapping on hard disk.

The latest Linux virtio balloon driver which KVM uses, operates somewhat differently comparing to the principle described above, in the matter that it combines ballooning and memory hotplug. The difference is that the guest operating system will notice the memory change. When the balloon "inflates", the "donor" virtual machine will notice a decrease in the total available memory as if a memory DIMM was removed and when the balloon "deflates" in the "beneficiary" it will notice an increase in the total available memory as if a memory DIMM was added.

KVM does not support automated ballooning, but even XEN Server and VMware that they do, they do it in a reactive way. Sudden reactive ballooning can lead to performance drawbacks because even if the virtual machine is trying to collaborate and give memory to the balloon driver, the memory might not be given to the host fast enough to avoid performance degradation. There is also a decision overhead introduced, while the hypervisor tries to find the ideal donor [15, 56].

2.3 Bayesian Networks

In this thesis, a Bayesian network will be trained to try to predict server load on virtual machines and adjust the physical resources they consume.

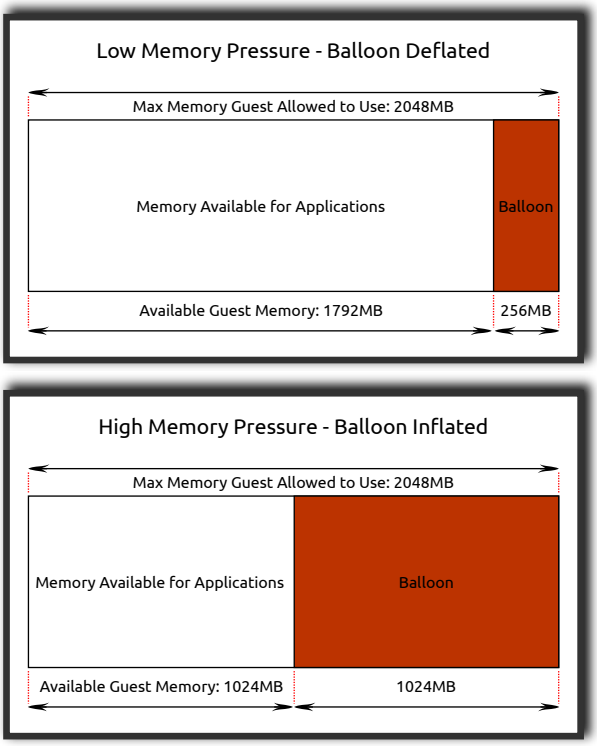


Figure 2.1: How ballooning works

2.3.1 How Bayesian Networks Differ

Most of the techniques trying to predict server utilisation use statistical analysis of collected system load data, to make a profile of each system and try to correlate it with the current conditions present in the system to foresee the future [8, 24, 25, 57, 58, 59].

These methods are proven to work, but they have a limited scope. They will work well for a specific task where they were adapted to be working with, but they cannot be easily expanded to correlate more information.

A Bayesian belief network is a probabilistic directed acyclic graphical (DAG) model, denoting the conditional dependence in a set of random variables embracing an ecosystem [27, 28]. Bayesian networks are popular to probabilistic artificially intelligent systems for their ability to learn from new observations [31]. A Bayesian network only needs some quantized data as inputs for its nodes and it will create a probability distribution function for each of them, affected by the causal relationships of the rest of the nodes around it. One of the highlights of Bayesian networks is that they are very robust when it comes to missing input data [60]. In this unpleasant but potential situation especially in research fields, the Bayesian network will make the best possible guess with the data which is known and in place.

Since the system is embodied by nodes and each node represents an input,

lots of independent inputs can be defined and correlated. This makes the network modular and easily expandable, as one could add more inputs on the run if new evidence appears to interfere with the system and the network will adapt and improve its results. Moreover, the longer the system runs it gives more fine grained results, as it keeps on learning from new observations of the existing inputs. With Bayesian networks, it would also be possible to combine results from statistical data or other predictive techniques used in other research works and get the most out of it [61]. The Bayesian network will use them in a similar manner as the rest of its inputs and mitigate its results. This makes the Bayesian network a very expandable wide purpose system, that it could be used for more than only predicting server utilisation.

2.3.2 Bayesian Inference

Bayesian inference is a method of statistical inference in which the posterior probability is a derivative of a prior probability and the likelihood function following the Bayes theorem (formula 2.4). Bayes theorem formula is derived by applying the conditional probability rule (2.1), a basic property of the intersection (set theory) (2.2) and again the conditional probability rule (2.3).

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{2.1}$$

$$P(A \cap B) = P(B \cap A) \tag{2.2}$$

$$P(B \cap A) \quad = \quad P(B|A) \cdot P(A) \tag{2.3}$$

$$P(A|B) = \frac{P(B|A)}{P(B)} \cdot P(A) \tag{2.4}$$

Bayes theorem gives a relationship between the probabilities of A ($P(A)$) and B ($P(B)$) and their conditional dependence of A given B ($P(A|B)$) and B given A ($P(B|A)$) [62]. In this equation $P(A|B)$ represents the posterior probability, while $P(A)$ is the prior probability and $P(B|A) \div P(B)$ is the rate which B affects the probability in A. $P(B|A)$ is the likelihood function. Bayesian inference is the basis of Bayesian networks.

2.3.3 More on Bayesian Networks

A Bayesian network encodes real life information and essentially imitates the way human intelligence works; Tell what you have to tell based on your training and experience by observations. It needs to be trained by a knowledgeable

person in the environment that it will have to work with, to provide better quality results.

To give a simple explanation, suppose of a newly born child. The child does not have any prior knowledge when it came to life. As the child grows, it learns from causes and its surrounding environment. When it goes to school it learns from teachers and when it becomes an adult and goes to the university it learns from the professors specialised in the subject of studies.

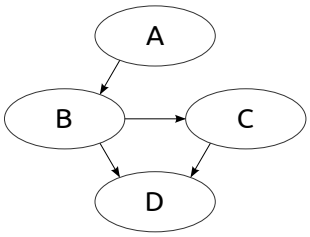
In this example wherever "child" or "adult" put the Bayesian network, wherever "teachers" or "professors" put the trainer of the Bayesian network, and wherever "causes" put the self learning capabilities of the Bayesian network.

When the child is very young, it might fall down and hurt itself for the very first time. It gets an unpleasant feeling out of this unknown situation to it. As the child grows and keeps on playing and falling and hurt itself, it gets again this unpleasant, yet known feeling now. As this event repeats while the child is growing, the uncertainty that when hurting itself it gets an unpleasant feeling becomes quite certain, so it tries to protect itself more. This is a description of a self learning causal procedure.

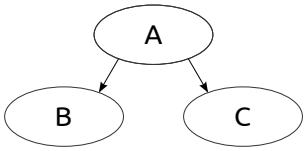
When the child goes to school, it is obvious that if the teacher is not a real teacher the child will not get proper education. Additionally, if an adult goes to a university to study computer science without any prior experience and there are no computer science qualified professors available to teach, the learning will be very poor so it will not be able to stand equally against another one who had proper education in computer science. This is an example why a Bayesian network trained by an expert in the field, will behave and give better results than if it was not trained by one with sufficient knowledge.

Bayesian networks are composed of nodes of random variables, which reflects characteristics of the system and they are chosen by the trainer. The causal relationship between nodes is defined using directed arrows from the cause node to the effect node and no loops are allowed to the DAG as shown in figure 2.2a [63]. Each node represents a probability distribution function and the Bayesian network will provide the posterior joint probability, as calculated by the Bayesian inference (2.4). This posterior probability represents the probability for the event represented by a node to happen, given the prior probabilities of it parent nodes and itself. The more expertise the trainer has on the field, the better the belief network will become eventually as the nodes are more accurately chosen and their relationships will be better tuned. More nodes for more accurate results can be added to the Bayesian network if there is evidence of interference from other sources and the current ones can adapt to behavioural changes of the system giving accurate results as more knowledge is absorbed. This means that the longer the system works, more confident results will be provided.

The nodes connected directly with an arrow, thus having direct impact to each other are conditionally dependent, while the rest of the nodes are conditionally independent of each other. As in figure 2.2b, Nodes B and C (effect nodes)



(a) An arrow from C back to A is not allowed as it would create a loop, but there could be another arrow from A to C.



(b) Nodes B and C are conditionally dependent with A, but they are conditionally independent between them.

Figure 2.2: Bayesian directed acyclic graphs

are conditionally dependent on their parent node A (cause node), while they are conditionally independent between them [63]. With this in mind, one can say that $P(B|A,C) = P(B|A)$ and $P(C|A,B) = P(C|A)$ and consequently the resulting joint probability function for this Bayesian network is:

$$P(A,B,C) = P(B|A) \cdot P(C|A) \cdot P(A) \tag{2.5}$$

To make a generalized form of (2.5) for any given number of nodes $X = X_1, X_2, ..., X_n$, the joint probability function for any given Bayesian network is:

$$P(X) = \prod_{i=1}^{n} P(X_i| \cap Parents(X_i)) \tag{2.6}$$

2.3.4 A Practical Example

In figure 2.3 a simple Bayesian network has been created and evaluated with SamIam, to get a clear understanding of the usage and philosophy behind a system like this [64]. SamIam is a free piece of software that as the official web site states, is *"a comprehensive tool for modeling and reasoning with Bayesian networks, developed in Java by the Automated Reasoning Group of Professor Adnan Darwiche at UCLA"*. SamIam includes a graphical user interface to let the user design a Bayesian network and a reasoning engine for the computation of the Bayesian inference.

This Bayesian Network, represents a very simple case of a fictional telecommunications company in which they monitor one of their text messaging (SMS) servers and they want to forecast its network traffic and memory utilisation. The company collected data over the years including timestamps, network and memory data for the targeted server. After an analysis of the data and based on their current experience, they have observed that **every year in the new year's eve**, people send so many text messages that their system is always struggling to serve. They have also observed that in **typical weekdays** the people send more text messages and that the **network traffic, significantly**

affects the memory of the server. The bold text marks some important keywords in the description of the problem. All of this information is encoded in the following Bayesian Network (figure 2.3) and the probability tables of each node of it (2.1, 2.2, 2.3, 2.4).

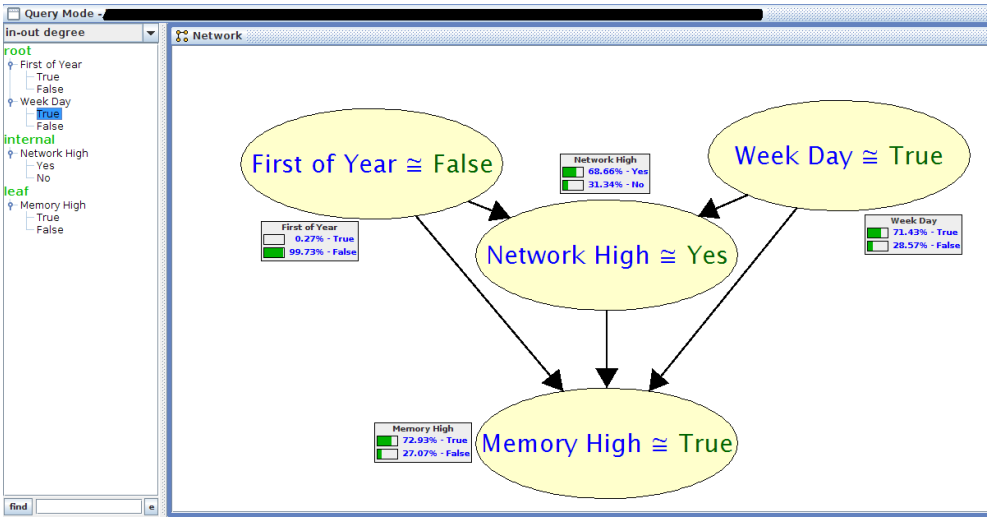


Figure 2.3: Simple Bayesian network to find the probability of high Network Usage and Memory, depending if it is a weekday or the first day of the year.

First of Year	
T	F
0.002739726 *(1/365)*	0.997260274 *(1-(1/365))*

Table 2.1: Example 2.3: Probability to be the First Day of The Year

Week Day	
T	F
0.7142857143 *(5/7)*	0.2857142857 *(1-(5/7))*

Table 2.2: Example 2.3: Probability to be a working Week Day

This real life information needs a way to be described to the Bayesian network and this is far simpler than other statistical methods involve rigorous mathematics. First the important keywords marked in bold letters in the previous description have to be taken into consideration and out of these, the nodes of the system with their conditional dependence and their descriptive states will be created which in this case they are 4 boolean state nodes.

The boolean nodes, thus only two allowed states (*True* or *False*) for all of the nodes have been used in this example for simplicity, but in a real scenario many states can be defined. Each node needs to get some initial probabilities (the so called prior probabilities) for each of its possible states, which they will be learned from previously observed data (the system needs to learn its working environment) and knowledge of the trainer.

		Network High	
First of Year	Weekday	T	F
T	T	0.999	0.001
T	F	0.999	0.001
F	T	0.8	0.2
F	F	0.4	0.6

Table 2.3: Example 2.3: Probabilities to get High network utilisation depending on if it is a weekday and the first of the year

			Memory High	
First of Year	Weekday	Net High	T	F
T	T	T	0.99	0.01
T	T	F	0.99	0.01
T	F	T	0.99	0.01
T	F	F	0.99	0.01
F	T	T	0.9	0.1
F	T	F	0.6	0.4
F	F	T	0.75	0.25
F	F	F	0.25	0.75

Table 2.4: Example 2.3: Probabilities to get High memory utilisation depending on if it is a weekday, the first of the year and if high network utilisation has been observed

"New year's eve" encoded as node *First of Year* and represents if it is January, 1st.

"Typical weekdays" encoded as node *Week Day* and represents if the day is between Monday and Friday.

"Network traffic" encoded as node *Network High* and represents if the network utilisation is more than 0.5 Gbps.

"Memory" encoded as node *Memory High* and represents if the memory utilisation is more than 80% for the text messaging server.

The *First of Year* and *Week Day* are root nodes, *Network High* is a child node of *First of Year* and *Week Day*, while *Memory High* is a child node of all of the other nodes. The network was designed like this because from the description, it is obvious that the date plays the initial role for text messaging and network utilisation is also playing an important role on memory utilisation.

In more detail on how the probabilities assigned to each node occurred, the *First of Year* node (table 2.1) can get a *True* value 1/365 = 0.002739726 of the times, while the probability to get a *False* value is 1 – (1/365) = 0.997260274. The probability node *Week Day* (table 2.2) gets a *True* value 5/7 = 0.7142857143 of the times, while it gets a *False* value only during the weekends, thus a 1 – (5/7) = 0.2857142857 of the time. These two nodes got their prior probabilities

hard-coded by the trainer, as their value will be fixed (every week will always have 5 out of 7 weekdays and there will be only once per year January, 1st). On the other hand, the probabilities of the network and memory utilisation are based on quantized collected data. For network utilisation (table 2.3), it has been observed by the logged data that no matter what the day is (weekday or weekend), if it is the first of the year it is 99.9% certain that the network traffic will be more than 0.5 Gbps. For the rest of the year, if it is a weekday there is 80% probability that the network utilisation will be higher than 0.5 Gbps, while for the weekends there is only 40% probability that the network traffic will surpass 0.5 Gbps. These probabilities are variable and they can change as the system is learning from newly collected data. A similar interpretation can be given for the memory utilisation and table 2.4.

Now the Bayesian network is trained and ready to give some results. Today it is Sunday so the company knows with 100% probability that tomorrow is Monday and it is not the 1st of January. This is given as known information to the Bayesian network, and the system will give the likelihood to get more than 80% memory utilisation and more than 0.5 Gbps of network traffic which in this case it is 84% and 80% respectively as shown in figure 2.4. Now when Monday comes, the company observes that it is not a busy day and the network utilisation is less than 0.5 Gbps so they add this knowledge to the system. The system then improves its results for the only unknown node which is the memory usage and it reports that there is 60% probability to get more than 80% of memory usage as shown in figure 2.5.

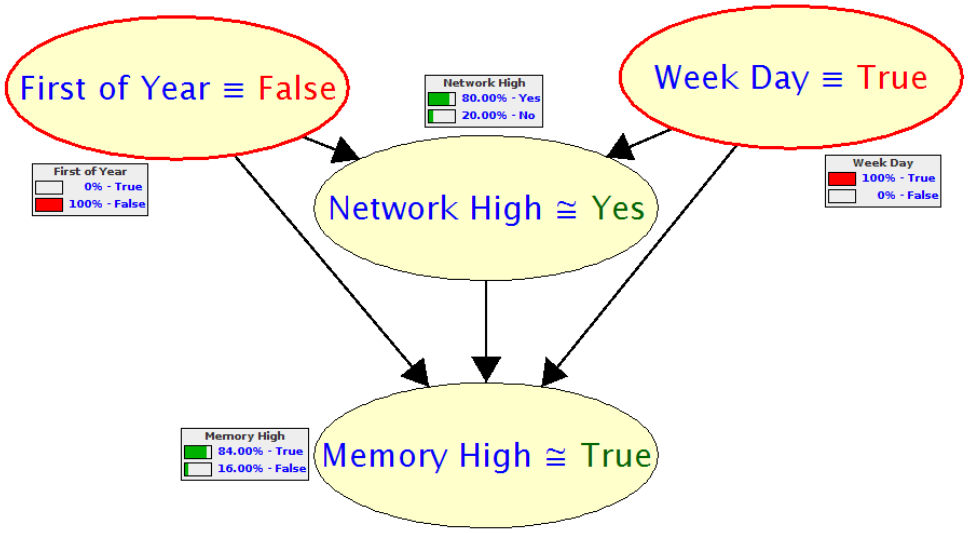


Figure 2.4: It is a weekday and it is not January, 1st

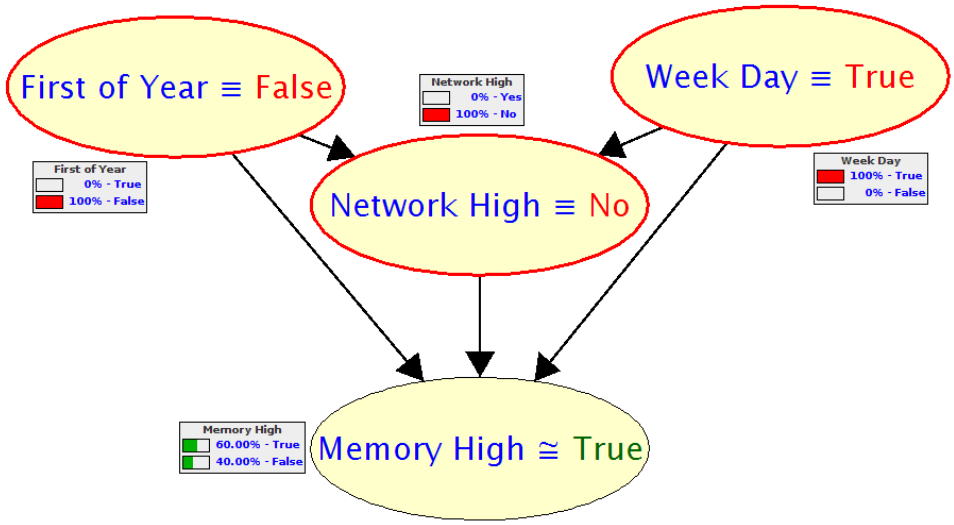


Figure 2.5: It is a weekday, it is not January, 1st and the network utilisation is less than 0.5 Gbps

2.3.4.1 Calculations

The Bayesian network will use the Bayesian theorem (equation 2.4) and the prior probabilities of each node as in the tables 2.1, 2.2, 2.3, 2.4, to calculate the posterior probabilities and if there are unknown variables it will use the law of total probability to calculate the joint probability of the network. Let the finite sample space $S = \{A_1, A_2, ..., An\}$, then for every B belonging to the probability space S, the law of total probability applies as illustrated in equation 2.7. If the conditional probability (equation 2.3) is applied to equation 2.7, then equation 2.8 occurs.

$$P(B) = \sum_{i=1}^{n} P(B \cap A_i) \tag{2.7}$$

$$P(B) = \sum_{i=1}^{n} P(B|A_i) \cdot P(A_i) \tag{2.8}$$

Now for the calculation of the likelihood that memory usage will be more than 80% given the weekday and if it is the first day of the year, while the network traffic is also unknown on the scenario demonstrated in figure 2.4, the Bayesian network will calculate its results as in equation 2.9. Where m, n, f, w denotes the initial letter of corresponding nodes and where X_T or X_F (replace X with any of the letters m, n, f, w) denotes if the state of this node is True or False to choose the correct prior probability.

$$P(m_T|f_F, w_T) = \frac{P(m_T, f_F, w_T)}{P(f_F, w_T)} = \tag{2.9}$$
$$= \frac{\sum_{n\in T,F} P(m_T, n, f_F, w_T)}{\sum_{m,n\in T,F} P(m, n, f_F, w_T)} =$$
$$= \frac{P(m_T, n_T, f_F, w_T) + P(m_T, n_F, f_F, w_T)}{P(m_T, n_T, f_F, w_T) + P(m_T, n_F, f_F, w_T) + P(m_F, n_T, f_F, w_T) + P(m_F, n_F, f_F, w_T)}$$

Here (equation 2.9) by applying the chain rule of probability (equation 2.10), the calculations end up with a product of Probabilities which is easy to substitute using the tables of knowledge of the system 2.1, 2.2, 2.3, 2.4 and eventually come to equation 2.11.

$$\begin{aligned} P(A_1, A_2, ..., A_n) &= P(A_1|A_2, ..., A_n) \cdot P(A_2|A_3, ..., A_n) \cdot \\ &\quad \cdot P(A_3|A_4, ..., A_n) \cdot ... \cdot P(A_n) \Rightarrow \\ P(\cap_{k=1}^{n} A_k) &= \prod_{k=1}^{n} P(A_k| \cap_{j=1}^{k-1} A_j) \end{aligned} \tag{2.10}$$

$$\frac{(m_T \cdot n_T \cdot f_F \cdot w_T) + (m_T \cdot n_F \cdot f_F \cdot w_T)}{(m_T \cdot n_T \cdot f_F \cdot w_T) + (m_T \cdot n_F \cdot f_F \cdot w_T) + (m_F \cdot n_T \cdot f_F \cdot w_T) + (m_F \cdot n_F \cdot f_F \cdot w_T)} = \tag{2.11}$$
$$= \frac{f_F \cdot w_T \cdot (m_T \cdot n_T + m_T \cdot n_F)}{f_F \cdot w_T \cdot (m_T \cdot n_T + m_T \cdot n_F + m_F \cdot n_T + m_F \cdot n_F)} =$$
$$= \frac{(1/365) \cdot (5/7) \cdot (0.9 \cdot 0.8 + 0.6 \cdot 0.2)}{(1/365) \cdot (5/7) \cdot (0.9 \cdot 0.8 + 0.6 \cdot 0.2 + 0.1 \cdot 0.8 + 0.4 \cdot 0.2)} = 0.84 = 84\%$$

The result tells that when it is not January, 1st and it is a weekday given that the network traffic is unknown, there is 84% probability that the memory usage will surpass 80% of the total memory available in the text messaging server of the company. Known that the nodes in this example are boolean, the probability that the memory usage will not surpass 80% of the total memory available in the server is 1 – 0.84 = 0.16 = 16%. Of course this could also be calculated in a similar manner with the example as the rest of the nodes as well.

The scenario in 2.5 is much simpler as there is only one unknown node and the results can be extracted only by looking the table of *Memory High* in 2.4.

2.3.5 A Solution for a N-P Problem

As depicted by the previous detailed example, the calculations for a very simple Bayesian network can become substantially demanding (take a look at the fractions in equations 2.9 and 2.11) as the system needs to calculate the belief for every single node. For a system with n boolean random variables, the

complete distribution is specified by 2^{n-1} joint probabilities [27]. If the system in the example was not using boolean variables, the total joint probabilities to be calculated would increase exponentially even for a system with 4 nodes like this. This is described as a NP hard mathematical problem, as the time for computation in large Bayesian networks with hundreds or thousands of nodes might be significant [27, 65, 66, 67, 68, 69]. Many approaches have been proposed to solve this computational challenge, with most of them suggesting heuristics and metaheuristics (genetic algorithms, swarm intelligence etc) to change the Bayesian network structure and avoid a greedy search through all of the nodes [65, 66, 67, 68, 69]. The main difference of heuristics and algorithms is that the algorithm will give the best possible answer by searching for every possible combination for a function, while Heuristics will give a much faster one but without any guarantee that this is the best one (most likely it will not). Sometimes it is infeasible to use algorithms due to the lack of computing power and then heuristics apply. The heuristics will make a much faster scatter search and try to discover a local maximum of the given function. This local maximum might happen to be the global maximum but only by chance. Sometimes it might be a very bad guess, so metaheuristics apply in a case like this. Metaheuristics will try to iteratively discover a better solution (if one exists) than the one found using heuristics.

2.4 Related Work

Research in the section of virtual machine dynamic resource allocation is highly active, as this is the key to achieve proper resource utilisation, environment friendliness, cost reduction and increased profits by physical resource oversubscription. There are so many papers found addressing the same topic, but a summary with the most closely related work with similar scope as this thesis is gathered in this section.

2.4.1 Adaptive Control of Virtualized Resources

Adaptive Control of Virtualized Resources in Utility Computing Environments [70].

This system is a quite simple approach based on control theory, for dynamically allocating virtualization resources in a single hypervisor to succeed on CPU QoS. The experiment comprises input data from the test virtual machines. An actuator in the control system controls the CPU scheduler of the hypervisor, to assign resources to the virtual machines so that they will not exceed 100% utilisation and performance decrement. They only focus on CPU utilisation.

2.4.2 MEmory Balancer (MEB)

Dynamic Memory Balancing for Virtual Machines [58].

MEmory Balancer (MEB) is a software which dynamically monitors the memory usage of virtual machines, it will then calculate its memory needs, and periodically reallocates memory to the virtual machines.

MEB has an estimator and a balancer. The estimator will build a Least Recently Used (LRU) memory pages histogram for each virtual machine and it will also monitor their swap space usage. Then the balancer runs with an interval and adjusts virtual machine memory, based on the information given by the estimator and the available host resources. Ballooning is used for the memory management from MEB.

2.4.3 Memory Buddies

Memory Buddies: Exploiting Page Sharing for Smart Colocation in Virtualized Data Centers [57].

Memory Buddies, is a project which will try to maximize the efficiency of the page or memory sharing feature available in hypervisors, between multiple hypervisors. The page sharing feature will only merge memory pages of virtual machines running in the same hypervisor, so in a data center with multiple hypervisors, memory sharing opportunities may be lost because the guest machines holding identical pages are located on different hosts.

The contribution of this paper is a memory fingerprinting technique to identify guests with high memory sharing potential. Then it will use live migration to move these hosts and shutdown or start on demand hypervisors not in use to trim down operational costs by reducing the energy consumption. Their evaluation shows a 17% increase of the virtual machines running in a data center.

2.4.4 Overdriver

Overdriver: Handling Memory Overload in an Oversubscribed Cloud [59].

Ultimate target of this project is to maximize competitiveness and profitability of cloud providers by oversubscribing customers. Since most of the physical resources are leased using virtual machines, oversubscription means that the cloud provider sells more subscriptions, or more virtual machines to their customers than what its infrastructure can actually handle if all of them would run at peak load at the same time. The coherence is that the peak load for each customer is largely transient (up to 88.1% of overloads last for less than 2 minutes) and not at the same time for all of them. Overdriver focuses on memory oversubscription as memory is not largely oversubscribed in practice

like CPU, because memory overload leads to swapping and consequently to severely degraded performance.

Existing methods to handle memory overload use mostly migration to another physical machine that can handle the memory requirements, but virtual machine migration is a heavyweight process needs also high network usage, best suited to handle sustained or predictable overloads. They propose a new application of network memory to manage overload giving it the name "cooperative swap". This will take swap pages and store them to memory servers over the network. Then they present Overdriver, the system that it will respectively choose between VM migration and cooperative swap to manage either sustained or transient overloads. Overload will create a probability profile for each virtual machine to decide after how much time the coming load is considered to be sustained load for the specific guest. When the increased memory load comes, it will first use its cooperative swap feature and if the load surpass the sustained threshold set for this guest (based on the probability profile), a live migration will be executed. Overdriver is a reactive tool to excessive memory load.

2.4.5 Managing SLA Violations

Dynamic Placement of Virtual Machines for Managing SLA Violations [8].

In this paper, the authors introduce a management algorithm for dynamic resource allocation in virtualized environments using live migration. Their elemental goal is to meet the Service Level Agreement (SLA) the company has with its customers, while reducing the operational costs from the data centers of the company. The algorithm in this work will measure logged data for each of the virtual machines and forecast the future demand. It will then remap guest virtual machines to different hosts. Their forecasting technique involves some statistical analysis to determine the type of resource usage of the guests and then classifies them to three categories:

1. Guests without variability or periodic behaviour. (No need to migrate)
2. Guests with slight variability and periodic behaviour. (Potentially good guests for migration)
3. Guests with strong variability and periodic behaviour. (Highest migration potential)

From those three categories the second and the third category of virtual machines are potential candidates for live migration. The first one shows sustainable resource usage so it does not need to change host often.

The next step is the calculation of available physical resources needed to handle the predicted load and start or shutdown non needed servers to reduce costs. This method is a proactive probabilistic method like the one proposed in this thesis but they use different statistical tools, while their experimental

studies are only focused on CPU utilisation which is known that it is much easier to handle since it is a time shared resource.

2.4.6 VMCTune

VMCTune: A Load Balancing Scheme for Virtual Machine Cluster Based on Dynamic Resource Allocation [12].

This paper proposes VMCTune, a tool that monitors the resource utilisation of virtual machines and their hosts. Then it uses dynamic resource allocation for virtual machines running on same hypervisor to achieve better local resource utilisation and if the QoS cannot be met, it uses live migration for virtual machines among different hypervisors to achieve global load balancing. The tool focuses in Paravirtualized machines and offers a reactive solution dealing with CPU, memory and network bandwidth. Paravirtualized machines are easier to handle their resources in comparison with the Fully virtualized ones, as they are aware of running in a virtualized environment.

VMCTune has several tools to monitor, log the data, schedule the resource allocation or issue a live migration if needed and a command line tool by which the user can monitor the status of the virtual machines and control the host.

Improving of scheduling algorithm of the live migration is the key work mentioned as their future research work in this paper, as the one used is a very simple best-fit algorithm. For example, if a virtual machine needs more resources and it cannot fit to the current hypervisor, it will be transferred in a different one with the available resources. There will be no attempt to squeeze resources from other hosts to achieve the best possible utilisation.

Chapter 3

Methodology

The methodology chapter will guide the reader through the system design, tools, analytical procedures used in this thesis, and the planned workflow to achieve the final goal. This is the successful prediction of memory load using Bayesian networks and use of this information to provide dynamic memory allocation.

3.1 System Design

In order to improve working efficiency the system is designed in a modular manner. Modularization is the separation process of parts serving different goals in a project. This leads to increased initial build effort, but it improves efficiency when adding new features and decreases complexity when a part needs to be substituted for any reason. It might be that something went wrong and needs to be replaced, or simply that one needs to test different modules to provide some modified functionality. Since this is an investigative thesis and different tests need to be made, the modularization helps a lot throughout the accomplishment of the task.

Five (5) modules have been defined for this project:

1. Virtualization Layer
2. Data Generation
3. Data Collection
4. Prediction
5. Final Action (Dynamic Memory Allocation)

From the 5 mentioned modules, an existing solution (libvirt [71]) is used for the virtualization layer, while the rest are manually implemented from the ground up.

As illustrated in the conceptual system design in figure 3.1, everything runs on top of a single physical machine which is the hypervisor. The hypervisor is split in two logical partitions. First is the space available for the virtual machines (left part on the figure) and the latter (right part on the figure) is the space available for the administration of the virtual machines.

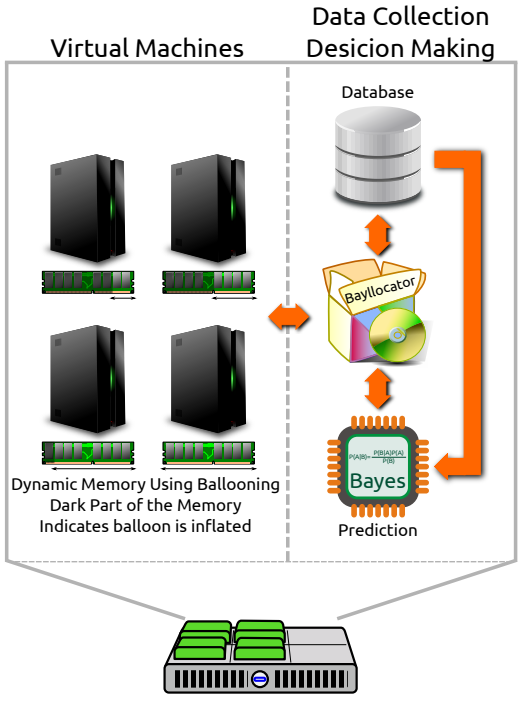


Figure 3.1: Conceptual system design

The 5 modules mentioned earlier can be spotted in the figure and associated as follows:

Data generation is taking place on the virtual machines. Data collection lies on the administrative part of the system together with the prediction system. A script collects the data through the virtualization layer (the dotted separation line) and stores it in a database (MySQL). The prediction then is made using as input the data from the database and the dynamic memory allocation mechanism is using this information to reallocate the memory using the virtualization layer. All of these different parts will be explained in detail later on in the methodology and result chapters.

3.1.1 Experiment Design

The infrastructure schematic already explained briefly in figure 3.1. On top of this, a test environment is being built for the experimental part of the project. Three virtual machines are created, and on each one of them memory load is generated and collected. Simple Bayesian networks are tested and in the end Bayllocator is created to put everything together. With the final system built, a simulation is running against real data collected. Bayllocator is the name given to the software created to accomplish the experiments in this thesis. It is a set of scripts for data collection, predictions, email notifications when the system is malfunctioning and dynamic memory allocation for multiple virtual machines using ballooning. It has been tested and works effectively on a KVM hypervisor. The important scripts of Bayllocator are explained later in the methodology and results chapters, and when Bayllocator is mentioned from now on, the reference is mainly to the main script which is responsible for the main task which is the dynamic memory allocation.

3.2 Hardware and Software

The hardware granted to proceed with the development and experiments is a powerful Dell R6 server with the specifications given in table 3.1. Because the server has 24GB of RAM, the memory available for the virtual machines was soft limited to less than 3GB to create some pressure.

Dell R6 Server	
CPU	2x Intel(R) Xeon(R) CPU E5530 @2.40GHz
Cores	2x 4 cores per CPU = 8 cores total
Memory (RAM)	12x 2GB = 24GB total
Hardware RAID	Symbios Logic SAS2008 PCI-Express Fusion-MPT SAS-2
Hard Disks	2x Dell ST9146803SS 73GB 10K RPM in RAID 0 setup for improved performance

Table 3.1: Hardware specifications

The operating system installed is Ubuntu Server 11.10 (Oneiric Ocelot) 64 bits, which it was the latest stable version of Ubuntu available in the beginning of the project.

3.2.1 Programming Languages

Perl and R are the chosen programming languages to write the program. Perl is simply chosen because it is the language the author is more familiar with and libvirt bindings have already been developed for it (read on section 3.2.2 for libvirt information). R on the other hand is a powerful free and open source statistics programming language with packages to implement Bayesian networks and probability propagation. Software which is specifically designed for Bayesian networks exists (SamIam [64] and Hugin[1] to mention a couple of them), but the free availability and ability of R to be automated and scripted makes it the software of choice. R is also used for the statistical analysis and all of the plotted charts. Most of the scripts produced for the needs of this thesis can be found in the appendix section A for further studying, but some important ones tied with the methodology and results will be explained through the following sections and chapters.

3.2.2 Virtualization Platform

The virtualization platform chosen for the experiments is KVM (Kernel Virtual machine). The KVM package available by default in the repositories for Ubuntu 11.10 is qemu-kvm version 0.14.1, which is replaced by version 1.0.0

[1] http://www.hugin.com

installed from an external ppa repository [2]. This change is considered necessary as a result of the choice of using QEMU GuestAgent (qemu-ga) [3] which needs a minimum version of qemu-kvm 0.15.

QEMU GuestAgent will be used to allow the hypervisor to communicate directly with the guests. It simplifies the overall setup and gives a robust hypervisor-guest communication through a secure Unix socket. The data collection script running on the hypervisor will mostly use this feature to access files (/proc/meminfo for example) from within the virtual machines, process the information and store it in a database. The traditional way would be to use a typical server/client application, but this involves the presumption that each guest has at least one virtual Ethernet adapter and correct IP address settings (more complicated setup).

Libvirt has to be mentioned here as well, which is a hypervisor agnostic virtualization API (application programming interface) serving as the virtualization layer in this project. The configuration and administration of the virtual machines are not happening by sending direct commands to the hypervisor, but to the libvirt instead. This means that if a different hypervisor (XEN for example) supports live ballooning as KVM does, it will most likely work without any changes to the software written.

3.3 Data Generation

Controlled random data will be generated and mostly used to carry out the experiments. Data generation is considered to be necessary to proceed for a couple of reasons:

1. There was no prior data collected for the demands of the project to be supplied from the institution when the thesis started.
2. It is a good starting point when one is in need to explore something new and the required experience is missing. Generated data has a unique feature. It can be controlled so that one can make prior assumptions and expectations need to be confirmed by the results. Talking for this project, there was no prior involvement and experience on Bayesian inference and networks.

For data generation, a script is created (*loadsim.pl - Appendix A.2*) and a flow chart in figure 3.2 explains its working logic. The design of this script was made with the ambition in mind to create random data with controlled randomness by modifying the variation. The script will generate data given 3 parameters:

1. Max memory to occupy (given in MB). This parameter will set the maximum memory the script will try to claim. The passed memory is a rough

[2] https://launchpad.net/ ukplc-team/+archive/ppa
[3] http://wiki.qemu.org/Features/QAPI/GuestAgent

estimate.

2. Max threads to create. This parameter will start the given number of random consuming memory threads and all of them together will consume a maximum amount of memory given by the first parameter.
3. Max runtime of the script. This will control the time length the script will be generating data.

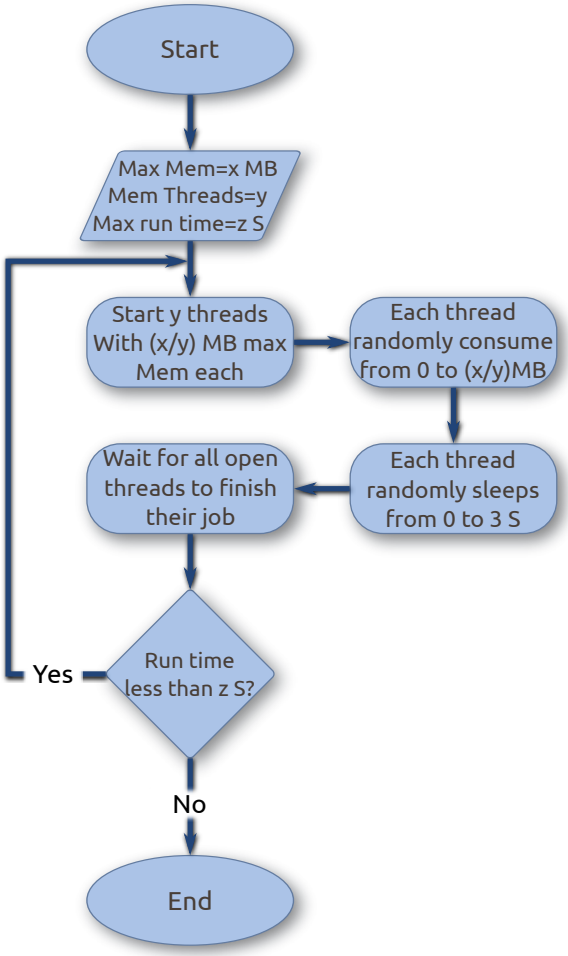


Figure 3.2: loadsim.pl - Script to generate data

The second parameter providing the number of threads to be started, is the one which it will control the variation of the data. More threads open, less variation in memory consumption and vice versa. Each thread will consume an amount of memory for a maximum time length of $3 \cdot 10^6$ microseconds (or 3 seconds) and all of the threads will wait for the other threads to finish until they will consume memory again in the next run.

To make this more understandable consider the following examples: We set the Max memory parameter to $1000MB$ and the number of threads to 1. The

program will start one thread and it will randomly consume from $0MB$ to $1000MB$ (uniform or flat random distribution) for a random time (flat random distribution as well) of 0 to 3 seconds. When this thread will finish its job, it will instantly start another one with the same characteristics and so on. In the first run the random memory to consume might be $949MB$ for $2.74s$, the next run will start instantly after $2.74s$ that the first run finished and it might randomly choose to consume $376MB$ for $1.31s$, next run $239MB$ for $1.03s$ and so on. It is obvious that the resulting memory consumption will be jumping from very low to very high values due to its high standard deviation value as explained with the following formulas and seen in figure 3.3a .

The randomly chosen values follow a uniform distribution, so 1 thread consuming from 0 to $1000MB$ has a probability density function as in equation 3.1:

$$f(x) = \begin{cases} \frac{1}{1000} & 0 \leq x \leq 1000 \\ 0 & elsewhere \end{cases} \tag{3.1}$$

Consequently this has an expected value given in equation 3.2, a variance given in equation 3.3 and a standard deviation given in equation 3.4.

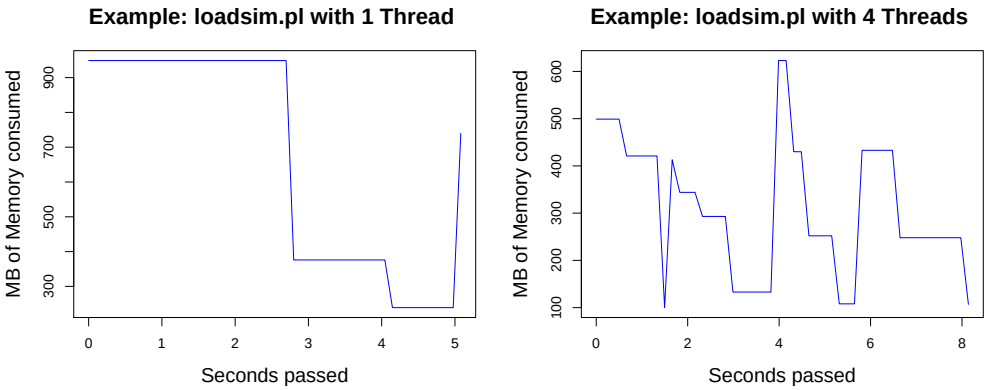
$$E(\overline{X}_n) = \mu = \int_0^{1000} x f(x) dx = \int_0^{1000} \frac{1}{1000} x dx = 500 \tag{3.2}$$

$$\begin{aligned} Var(X) = \sigma^2 \quad = \quad & \int_0^{1000} (x-\mu)^2 f(x) dx = \\ & \int_0^{1000} (x-500)^2 \frac{1}{1000} dx = \\ & \frac{250000}{3} \simeq 83333 \end{aligned} \tag{3.3}$$

$$\sigma = \sqrt{Var(X)} = \sqrt{\sigma^2} = \sqrt{83333} \simeq 289MB \tag{3.4}$$

In a different example with Max memory set to $1000MB$ and the number of threads set to 4, the script will start 4 threads that each one of them will consume up to $250MB$ ($4 \cdot 250MB = 1000MB$) and each of them will sleep from 0 to 3 seconds. The result will be the same mean value that was calculated in the example with 1 thread running, but with a much lower standard deviation (half in this case). A consequence of this, of course, is less variation. A sample chart with 4 threads can be seen in figure 3.3b and an analytical explanation in the following formulas.

$$f(x) = \begin{cases} \frac{1}{250} & 0 \leq x \leq 250 \\ 0 & elsewhere \end{cases} \tag{3.5}$$



(a) loadsim.pl example with 1 running thread.

(b) loadsim.pl example with 4 running threads.

Figure 3.3: loadsim.pl - Generated load examples

$$E(\overline{X}_n) = \mu = \int_0^{250} xf(x)dx = \int_0^{250} \frac{1}{250} xdx = 125 \tag{3.6}$$

Because 4 threads are started, the total mean or expected value:

$$E(\overline{X}_n)_{total} = 4 \cdot E(\overline{X}_n) = 4 \cdot 125 = 500 \tag{3.7}$$

$$\begin{aligned} Var(X) = \sigma^2 \quad = \quad & \int_0^{250} (x-\mu)^2 f(x)dx = \\ & \int_0^{250} (x-125)^2 \frac{1}{250} dx = \\ & \frac{15625}{3} \simeq 5208 \end{aligned} \tag{3.8}$$

Because 4 threads are started, the total variance is:

$$Var(X)_{total} = 4 \cdot Var(X) = 4 \cdot \frac{15625}{3} = \frac{62500}{3} \simeq 20833 \tag{3.9}$$

$$\sigma = \sqrt{Var(X)_{total}} = \sqrt{20833} \simeq 144MB \tag{3.10}$$

In figure 3.3b one can observe more small jumps from one memory consumption state to another and this is because of the random time each of the 4 threads sleeps. For example if in the first run the:

- 1st thread consumes $78MB$ of memory for $0.53s$.

- 2nd thread consumes $127MB$ of memory for $1.39s$.
- 3rd thread consumes $194MB$ of memory for $1.45s$.
- 4th thread consumes $100MB$ of memory for $1.66s$.

Then the summary of all threads which is $78 + 127 + 194 + 100 = 499MB$ will be consumed from $0s$ to $0.53s$. At $0.53s$ the 1st thread has finished its job and the summary of the 2nd, 3rd and 4th thread which is $127 + 194 + 100 = 421MB$ will be consumed from $0.53s$ to $1.39s$ and so forth. In this way, small local (per run) variations are introduced to simulate a more virtually real workload that the memory is changing as different system applications start, stop or asking for more or less memory when they run.

3.4 Data Collection and Storage

To be able to predict resource utilisation of a system, prior knowledge from observations for each of the running servers (virtual machines in this case) is needed. This entails data collection and storage of this data for further analysis from the predictive system which is the Bayesian network in this case.

A script (*collect-data.pl - Appendix A.4*) was created to deliver the function of data collection and storage. The flow chart in figure 3.4 gives a brief idea of how this script works and stores the data by iterating through the active virtual machines running on the hypervisor.

The data is stored into a MySQL database and once per day a second script (*data-smoothing.pl - Appendix A.5*) is running to provide rolling averages of the memory data. The default rolling average time length can be defined in the configuration file (*bayllocator.cfg A.18*) of the software or given as a command line argument to the script.

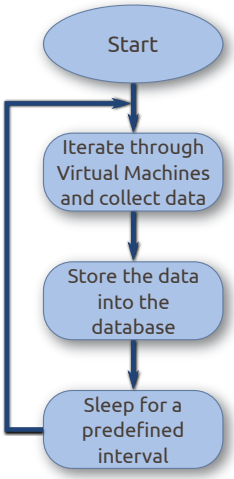


Figure 3.4: collect-data.pl - Script to collect data

3.4.1 Data Smoothing

The rolling averages technique is known as data smoothing and as nicely described in a book [72] it is used *to capture important patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena*. Here it is used to filter out extreme memory changes since many peak values appear due to the randomly generated data.

The effect of the smoothing applied on some generated data can be viewed in figure 3.5. The red line represents the random data and one can easily spot the high and low peaks. The blue line shows the 5 samples rolling averages of these data and it is much more smooth.

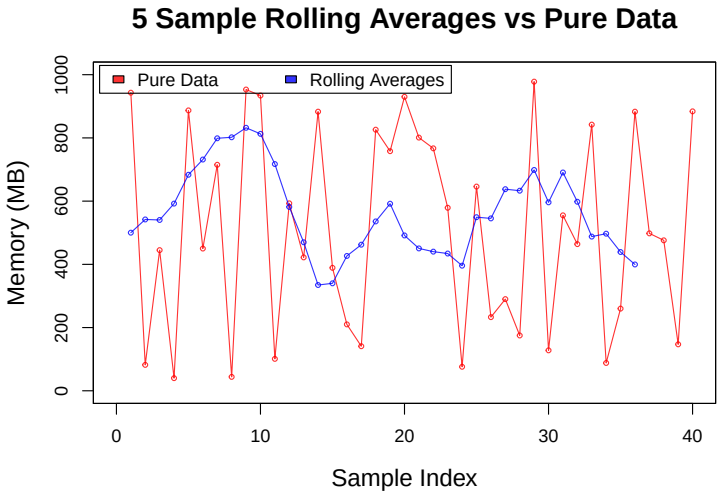


Figure 3.5: 5 Samples rolling averages applied in 40 samples

3.4.2 Data Collection from Real Servers

So far, only generated data and data collection from virtual machines in the testbed environment is described. This is a major part and the main path on the Bayesian network prediction design and study. Real life data from two servers in production (nexus2.iu.hio.no and studssh.iu.hio.no) is still collected in parallel; A simulated prediction based on this data will be run in the end, which will give an indication of how good the designed Bayesian network and its predictions work on real systems.

3.5 Bayesian Prediction

The application of a Bayesian network for server utilisation prediction is the core part and main research theme in this thesis. The focus is on the application itself and use of the output for dynamic memory allocation. Three existing reliable tools based on research are used (SamIam, bnlearn, gRain [64, 73, 74]) to build and evaluate the usefulness of different Bayesian networks in predicting computing workload.

3.5.1 Classification of the Predictions

Since working on prediction, some kind of classification is needed to be able to determine if the prediction is acceptable or not. The perfect prediction would be to get exactly what was predicted, although, very many times this is infeasible but a prediction does not need to be 100% accurate to be useful. A close prediction can still provide important information. The level of accuracy of what is considered "acceptable prediction" might vary a lot in different use cases and it always depends on the application and how critical will be the effect of the actions taken in case of a wrong prediction. In this project it is chosen to give an overall evaluation of the predictions by:

1. Plotting the predicted versus the observed data on the same chart for a quick visual comparison.
2. Plotting of the absolute difference which is the distance of the predicted versus the observed values.
3. Providing a boxplot with the median distance highlighted, a percentage of the outliers given the total predictions made and the median value of the distance of the outliers.

Outliers are considered to be the values lying very far from the lower (Q_L) and upper (Q_U) quartiles, where 50% of the total values around the median value rest. This is known as the interquartile range (IQR). The IQR multiplied with a constant κ, gives a new range r and values farther than $Q_L - r$ and $Q_U + r$ (check equation 3.11) are the calculated outliers. A constant value $\kappa = 1.5$ is used which is a commonly used value and the default value for the boxplot[4] in R statistics language which is used for the statistical analysis.

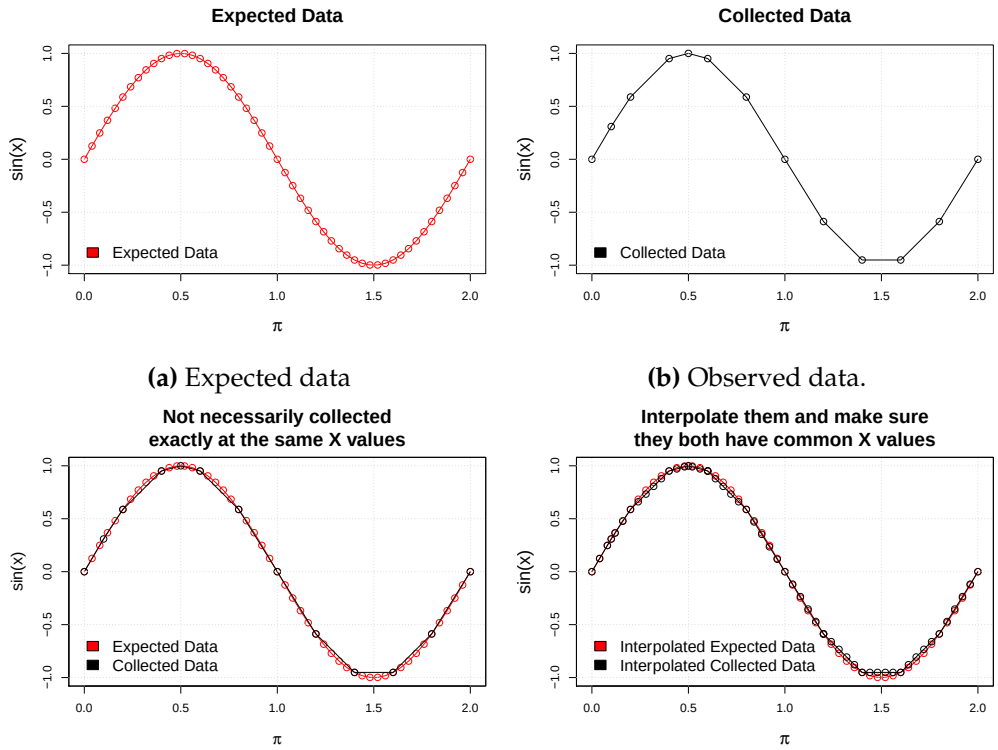
$$r = \kappa \cdot (Q_U - Q_L) \tag{3.11}$$
$$Outlier > Q_U + r \quad or \quad Outlier < Q_L - r$$

3.5.1.1 Data Interpolation

When a prediction is made, the timestamp for when this prediction was made for, is stored for later plotting and correlation with the observed data. Similarly when the data collection script is collecting data, it will store the timestamp of when the data was collected. Sometimes the timestamps might not coincide only for a few seconds, thus, for a given predicted timestamp there is no corresponding observation. This makes the direct comparison impossible as described in section 3.5.1 (by finding the absolute difference of the predicted versus the observed data). To bypass this issue, simple linear data interpolation is used [75]. Data interpolation is the technique of constructing new data points between existing data points. An example can be as demonstrated in

[4] http://stat.ethz.ch/R-manual/R-devel/library/graphics/html/boxplot.html

the figures 3.6 and the round marks on the figures represent the data points before and after the interpolation.



(a) Expected data

(b) Observed data.

(c) Data values mismatch.

(d) Interpolated data with added data points so that x axis values of the two data sets coincide.

Figure 3.6: Example: Linear data interpolation

3.6 Dynamic Memory Allocation

Dynamic memory allocation driven by the Bayesian predictions is the outcome of this thesis and short term prediction is the main goal. A memory allocation script will be created and it will try to reallocate the memory of the virtual machines 5 minutes in advance before the expected memory comes.

3.6.1 Evaluation of Virtual Machines Performance

To evaluate the usefulness of the dynamic memory allocation, simple workload for an application will be generated in excess of the generated data described in section 3.3. An Apache web server is running a PHP memory demanding script and the execution time of it is measured and stored in the virtual machine. Also a secure copy operation (SCP) is initiated from the hyper-

visor to the virtual machine to receive the stored file and the execution time for this transfer is measured as well. The tests will run for a long period of time, first without the ballooning and then with ballooning enabled. The results will be compared.

Chapter 4

Results

In the results chapter, a summarized outcome of the experiments and important scripts created is depicted. As described in section 3.1.1, three virtual machines have been setup, and on each one of them a script runs (check section 3.3) which generates CPU and memory load in different times of the day for different time lengths and different days of the week. This was one of the first tasks achieved and data collection started because it was expected that lots of data would be needed to feed the Bayesian network later on. In parallel with the data collection, simple Bayesian networks were tested to see how they behave and operate. This helped to get familiar with the working tools as well. A big effort was then made to apply the Bayesian network prediction on the collected data, improve it by trying different Bayesian networks and embed it to Bayllocator to provide the predictive dynamic memory allocation. With the final system built, a simulation was also ran against real data collected.

4.1 Virtual Machines Setup

The virtual machines were configured to be allowed to use a maximum of 4GB of memory and they boot up by default with 1GB. To setup the guest machines for communication with the hypervisor and data collection, the QEMU guest agent binary ("qemu-ga") has to be copied to each of the guests and it has to be made sure that two instances of it will start at boot time. The first instance is needed from the data collection script to collect and store the data in the database and the second is needed by the dynamic memory allocation script to read current guest system states affecting the predictions. One way to accomplish this is by adding the lines in code block 4.1 into the "/etc/rc.local" of the guest system.

Listing 4.1: Guest rc.local

```
1 /bin/qemu-ga -p /dev/virtio-ports/org.qemu.guest_agent.0 -d -f /var/run/qemu-ga-0.pid
2 /bin/qemu-ga -p /dev/virtio-ports/org.qemu.guest_agent.1 -d -f /var/run/qemu-ga-1.pid
```

4.1.1 Data Generation

The 3 guests are idle by default, so the *loadsim.pl* script was copied to all of them and it is running on cron with the following crontab lines for each guest:

Listing 4.2: Crontab line to generate system utilization - Test Server 1

```
1 0 8 * * 1-5 root /usr/bin/perl -e 'alarm shift @ARGV; exec @ARGV' 36000 /bin/loadsim.pl -M 1000 ↙
      ↳ -m 3 -c 3
```

Listing 4.3: Crontab line to generate system utilization - Test Server 2

```
1 0 23 * * * root /usr/bin/perl -e 'alarm shift @ARGV; exec @ARGV' 14400 /bin/loadsim.pl -m 2 -M ↙
      ↳ 600 -c 2
```

Listing 4.4: Crontab line to generate system utilization - Test Server 3

```
1 0 16 * * 5 root /usr/bin/perl -e 'alarm shift @ARGV; exec @ARGV' 3600 /bin/loadsim.pl -m 1 -M 300 ↙
      ↳ -c 1
```

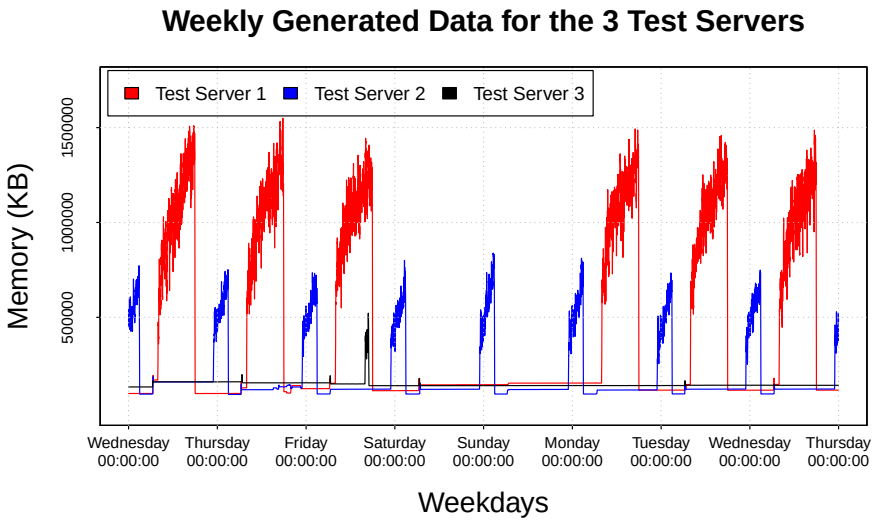


Figure 4.1: Typical weekly generated workload

As seen in crontab line for test server 1 at code block 4.2, the script will start generating load at 8am (08:00) from Monday to Friday and it will stop generating load after 36000 seconds (10 hours) has passed, at 6pm (18:00). It will consume a maximum of 1000MB of memory using 3 threads. Similarly for test server 2, 600MB of max memory will be consumed with 2 threads from 23:00-03:00 for every single day and for test server 3 only 300MB with 1 thread every Friday from 16:00-17:00. Server 1 in a real world example could represent a highly active server during working hours on weekdays, server 2 could represent a backup server taking backups every night and server 3 a not so active server increasing its load slightly once per week only for 1 hour.

The plot in figure 4.1 shows how a typical week of generated data looks like. The smoothed 5 minutes averages (explained in section 3.4.1) of the total memory in use on each guest is plotted. When talking about total memory in use on y axis (always from now on), it is the actual memory and swap in use observed, without any cached memory or cached swap involved.

4.2 Hypervisor Setup

The hypervisor is the main controller. It is responsible to run the virtual machines, the data collection script (*collect-data.pl - Appendix A.4*), the database, the script to create the rolling averages (*data-smoothing.pl - Appendix A.5*), the predictions and the main script (*bayllocator.pl - Appendix A.1*) which will dynamically allocate the memory.

The data collection script and the main script, needs communication with the guest machines using the QEMU guest agent through a "virtio-serial" interface. To simplify the setup procedure of the "virtio-serial" interface, the *add-unix-sockets-to-vm-conf.pl (Appendix A.9)* script has been created that it can be simply run as in the code block 4.5

Listing 4.5: Add unix sockets

```
1 $ add-unix-sockets-to-vm-conf.pl -v -d
2 Adding Unix Sockets to guest: UbuntuServer3
3 Adding Unix Sockets to guest: UbuntuServer1
4 Adding Unix Sockets to guest: UbuntuServer2
```

4.3 Prediction

R statistics bnlearn package and gRain were used to build the Bayesian networks. Bnlearn is a complete package to deal with Bayesian networks, but its main focus is on Bayesian network structure learning (which is a feature not used here) from data. It supports the extraction of the conditional probability tables (CPT) from data, but it is very slow on making queries. GRain focuses on probability propagation and it is much faster on querying a Bayesian network. Bnlearn was eventually used to define the Bayesian network structure manually and gRain is using this structure to extract the CPTs from the data and perform the queries.

4.3.1 Discrete Data

Learning the CPTs from data requires discrete values and good quantization of these values to reduce the number of states for each node without losing potentially important information. If the Bayesian network structure consists

from many nodes with many states each, the queries might be very slow to be acceptable.

The smoothed data is used for CPTs learning and it is converted to discrete values with a script (*data-to-bn-states.pl - Appendix A.6*) once every day (so every day the system improves its knowledge). A sample of how the converted data looks like follows in code block 4.6. Each column represents one Bayesian network node with its name as defined in the first row, and the rest of the rows contain quantized data. For example all of the discrete values when the memory is between 100MB and 200MB, will be represented with the state *mem_100_200*.

Listing 4.6: Discrete State for 5 nodes (each column represents one node)

```
1 W,H,M,CMU,FMU
2 Monday,h22,m55_59,mem_100_200,mem_100_200
3 Monday,h22,m55_59,mem_100_200,mem_100_200
4 Monday,h22,m55_59,mem_100_200,mem_100_200
5 Monday,h23,m00_04,mem_100_200,mem_100_200
6 Monday,h23,m00_04,mem_100_200,mem_100_200
7 Monday,h23,m00_04,mem_100_200,mem_100_200
```

The data in the previous example could be used as learning parameters for the Bayesian network in figure 4.2. Pay attention to the matching headers and the node names.

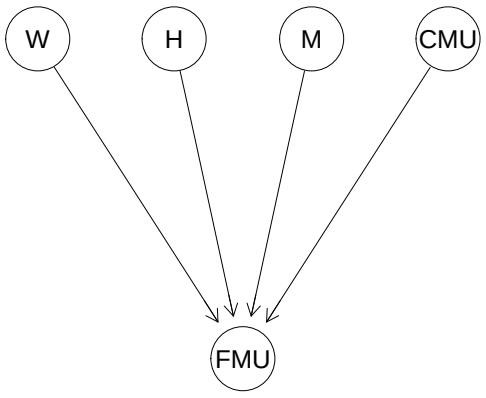


Figure 4.2: Sample Bayesian network with 5 nodes:

$$
\begin{aligned}
W: &\quad DayofWeek \\
H: &\quad HourofthеDay \\
M: &\quad Minutes \\
CMU: &\quad CurrentMemoryinUse \\
FMU: &\quad Future(Predicted)MemorytoUse
\end{aligned}
$$

4.3.2 Prediction with R Script

The Bayesian prediction is being made with the an R script (*QueryNet.R - Appendix A.10*) script which it will be given as command line arguments (see code block 4.7) the states of the known nodes, the name of the node which the answer is to be provided and the file with the discrete values to learn the CPTs from. The answer is a two columns comma separated output with the first column showing the possible state and the second column the probability (a percentage) to observe this state for the given inputs. In the example on the code block 4.7, the Bayesian network is asked "what is the expected memory when it is Monday, 07:55-07:59 and the currently observed memory in the system is between 100MB and 200MB". The answer is 14.8% to get 100MB-200MB, 24.3% to get 200MB-300MB and so on.

Listing 4.7: QueryNet.R sample execution

```
1 $ QueryNet.R /etc/bayllocator/UbuntuServer1.dat FMU Monday h07 m55_59 mem_100_200
2 mem_100_200,0.148
3 mem_200_300,0.243
4 mem_300_400,0.257
5 mem_400_500,0.226
6 mem_500_600,0.091
7 mem_600_700,0.026
8 mem_700_800,0.009
```

4.3.3 Choosing the Expected Memory

A fast Bayesian network (performance wise) needs a minimum possible number of nodes with the least possible discrete states as long as it serves the purpose of its design. Trying to follow this rule, the memory node of the first Bayesian network created was given in a percentage of the maximum memory. A node like this can adapt to any system without change (in need of additional states), as long as one knows the maximum memory of the corresponding system. This network introduced some problems later in the experiments and it had to be changed to absolute values similar to the example in the code block 4.7

4.3.3.1 Memory States in Percentage of Total Memory

The example in listing 4.7 shows the output of the prediction script for absolute quantized values. In the following example (code block 4.8) the memory is quantized in percentages of the total memory. The result from the Bayesian networks apparently contains more than one probabilities (the output of the script has more than one row), so a single number has to be chosen as the expected memory. To make a fair decision, all of the given results are used to calculate this number.

Listing 4.8: QueryNet.R sample execution

```
$ QueryNet.R /etc/bayllocator/UbuntuServer1.dat FMU Monday h07 m55_59 mem_100_200
  rate_0_10,0.168
rate_10_20,0.327
rate_20_30,0.384
rate_30_40,0.121
```

First as illustrated in equations 4.1, the mean percentage of the lower and higher percentages of each discrete memory states is computed. Then the expected memory for each row, known that the total memory is 1000MB for example is calculated as:

$$\begin{aligned} rate\_0\_10 &= (0.00 + 0.10)/2 = 0.05 * 1000 = 50MB \\ rate\_10\_20 &= (0.10 + 0.20)/2 = 0.15 * 1000 = 150MB \\ rate\_20\_30 &= (0.20 + 0.30)/2 = 0.25 * 1000 = 250MB \\ rate\_30\_40 &= (0.30 + 0.40)/2 = 0.35 * 1000 = 350MB \end{aligned} \tag{4.1}$$

These numbers are multiplied with the corresponding probabilities from the second column of the code block 4.8 and their summary is the expected memory.

$$\begin{aligned} 50 \cdot 0.168 &= 8.4 \\ 150 \cdot 0.327 &= 49.05 \\ 250 \cdot 0.384 &= 96 \\ 350 \cdot 0.121 &= 42.35 \end{aligned} \tag{4.2}$$

$$8.4 + 49.05 + 96 + 42.35 = 195.8MB$$

4.3.3.2 Memory States in Absolute Values

In section 4.3.3.1 an explanation is given on how to get a single value from quantized values given in a percentage of the maximum memory. As stated this technique introduced some problems so the example in 4.7, with quantized absolute values is used in the final Bayesian network and the calculations are very similar (check equations 4.3 and 4.4)

$$
\begin{aligned}
mem\_100\_200 &= (100+200)/2 = 150 \\
mem\_200\_300 &= (200+300)/2 = 250 \\
mem\_300\_400 &= (300+400)/2 = 350 \\
mem\_400\_500 &= (400+500)/2 = 450 \\
mem\_500\_600 &= (500+600)/2 = 550 \\
mem\_600\_700 &= (600+700)/2 = 650 \\
mem\_700\_800 &= (700+800)/2 = 750
\end{aligned} \tag{4.3}
$$

$$
\begin{aligned}
150 \cdot 0.148 &= 22.2 \\
250 \cdot 0.243 &= 60.75 \\
350 \cdot 0.257 &= 89.95 \\
450 \cdot 0.226 &= 101.7 \\
550 \cdot 0.091 &= 50.05 \\
650 \cdot 0.026 &= 16.9 \\
750 \cdot 0.009 &= 6.75
\end{aligned} \tag{4.4}
$$

$$22.2 + 60.75 + 89.95 + 101.7 + 50.05 + 16.9 + 6.75 = 348.3MB$$

4.4 Dynamic Memory Allocation Using Bayllocator

Dynamic memory allocation driven by the Bayesian predictions is the goal of this thesis. The main script that puts everything together is *bayllocator.pl (Appendix A.1)*. It is a quite complex script consists of almost 560 lines (around 500 without the comments). Some clarifications of the principles this script operates are given as bullets in this section and a very detailed flow chart which explains visually the entire operation steps can be studied in figure 4.3.

Bayllocator will:

- Make a prediction of how much memory each of the virtual machines will need and set this amount of memory plus some predefined percentage (default 15% more).
- Ensure that the virtual machines do not get less or more than the minimum and maximum limits set.
- Make sure that the hypervisor will not swap any memory by specifying a guaranteed amount of memory for the hypervisor using the *$config{hypervisor memory}* variable in the configuration file of Bayllocator (*bayllocator.cfg A.18*). In the case of high memory pressure, it is preferred that the virtual machines start swapping, instead of the hypervisor

which is responsible to handle them all. This amount was set to 21GB, leaving less than 3GB available to the guests to create some pressure.

- Distribute fairly (according to a percentage of the needed memory of the total predicted for the virtual machines) any excessive hypervisor memory to the virtual machines. If there is more memory than needed available, it can be used for disk caching purposes so it will boost guest performance.
- Claim memory fairly from virtual machines if they need more than the total amount allowed to get to avoid hypervisor swapping. In this case some of the virtual machines might start swapping under pressure.

4.4.1 How Memory is Allocated

The prediction described in section 4.3 is an expectation of the actual memory each of the virtual machines will need to use and not the memory the hypervisor will need to set. The reason is a memory overhead introduced by the hypervisor so more memory needs to be set.

4.4.1.1 Hypervisor Memory Overhead

When using memory ballooning, a fixed memory overhead related to the maximum allowed memory for each guest is introduced by the hypervisor. To find out how much is the memory overhead, a simple test was made. Two virtual machines were created with the maximum memory limit set to 4GB (4194304KB) for the first one and 8GB (8388608KB) for the second one. Both of them were booted up with 1GB of memory and their actual memory was changed with manual ballooning. The actual memory occupied in the hypervisor and the maximum total memory given to the guest were observed and a subtraction was made with the following formula:

$$HRM - GRM = HMO$$

$$\begin{aligned} HRM: &\quad HypervisorReportedMemoryPerGuest \\ GRM: &\quad GuestOSReportedMemory \\ HMO: &\quad HypervisorMemoryOverhead \end{aligned}$$

The results from the simple equations 4.5 for the first guest and 4.6 for the second guest shown that the overhead is fixed. A virtual machine with a maximum limit set to 4GB has roughly 135MB hypervisor overhead and a virtual machine with a maximum limit set to 8GB has almost 190MB overhead .

$$\begin{aligned} 1048576 - 910712 &= 137864 \\ 4194304 - 4056440 &= 137864 \end{aligned} \tag{4.5}$$

$$\begin{aligned} 1048576 - 853352 &= 195224 \\ 8388608 - 8193384 &= 195224 \end{aligned} \tag{4.6}$$

After the explanation, it is obvious that if the prediction is 800MB for a guest with a maximum limit of 4GB, the hypervisor will need to set at least the $PredictedMem + HypervisorOverhead$ which is $800MB + 135MB = 935MB$, or $800MB + 190MB = 990MB$ if the guest has a maximum limit of 8GB.

4.4.1.2 Some Extra Memory

The hypervisor memory overhead has been added to the prediction in the previous section 4.4.1.1, but the memory is just enough to serve the expectation of the virtual machine. If the prediction is exact and the virtual machine load reach close to the predicted memory, there is a high chance of memory swapping. To avoid this, +15% is added on top of the predicted memory. This percentage could be changed by the user but it should be balanced because the bigger this number, the safer for the virtual machine as it will get more memory, but it will consume more resources possibly needed by other guests. After the addition of the extra memory, a guest with a prediction of 800MB will get this amount of memory: $800 + (800 * 0.15) + 135 = 800 + 120 + 135 = 1055MB$

4.4.1.3 Respect the Limits

After applying the steps described in sections 4.4.1.1 and 4.4.1.2 for all of the running guests, either the hypervisor may not have enough memory to serve the demands for all of the guests or some of the guests might be violating their minimum and maximum allowed memory limits. For example a calculation of 4.5GB might have occurred for a guest with a maximum limit set on 4GB. The system will then remove this 0.5GB of memory in excess. In a case that the total (for all of the guests together) calculated memory needed is 4GB but the hypervisor has only 3GB available for the guests, then a percentage of how much memory corresponds to each guest from the total calculated memory is computed. Following this percentage, 1GB of total memory from all of the virtual machines will be removed to avoid hypervisor swapping, as shown in table 4.1.

4.4.1.4 Improve Performance if Extra Memory is Available

In a case that the total calculated memory is less than the available memory for virtual machines from the hypervisor, a similar procedure as explained in table 4.1 will take place. This time it will have the opposite effect of adding memory to improve guest performance instead of removing. More memory

Calculated Memory Per Guest			
Guest Name	Memory Calculated to be Set	Percentage of Total Calculated Memory	Memory to Set After Fixes
Guest 1	2GB	50%	$2GB - (0.5 * 1GB) = 1.5GB$
Guest 2	1.5GB	37.5%	$1.5 - (0.375 * 1GB) = 1.125GB$
Guest 3	0.5GB	12.5%	$0.5 - (0.125 * 1GB) = 0.375GB$

Table 4.1: Calculated memory per guest to respect the memory limits

can be used for disk caching reasons even if it is not needed by applications running in the guest.

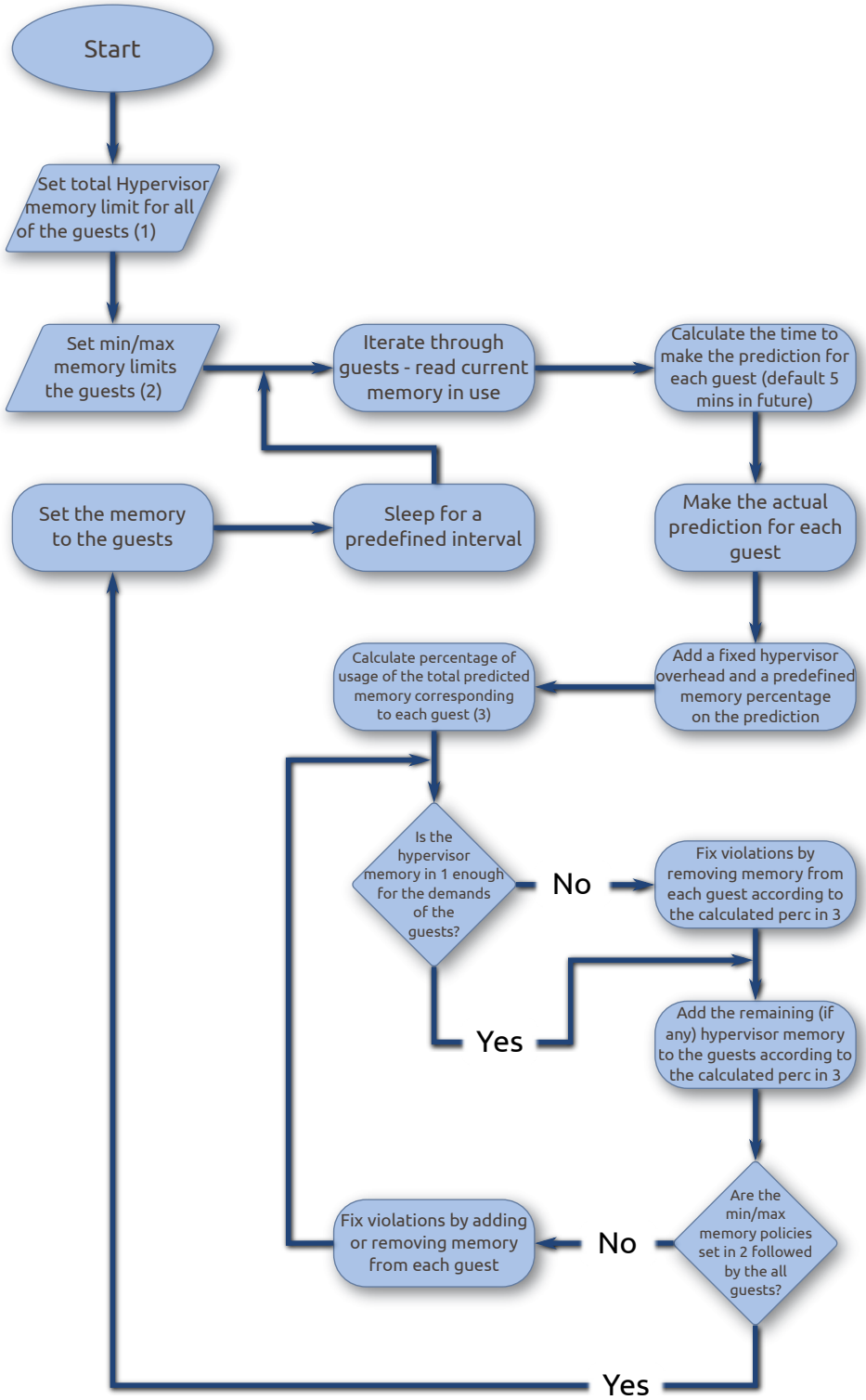


Figure 4.3: bayllocator.pl: Main script to dynamically allocate memory using the predictions

Chapter 5

Analysis

In the analysis chapter a comprehensive evaluation of the results will take place. The Bayesian networks tested are explained in detail and their results are analysed both for the test environment and the data collected from real servers. The positive impact in performance of the dynamic memory allocation driven by Bayllocator is also displayed. Whenever the word "ballooning" or the phrase "dynamic memory allocation" is referred in this chapter, means that Bayllocator is behind.

5.1 Bayesian Networks Evaluation

The following table (5.1) expands the abbreviations of the node names for the Bayesian networks following in the coming sections.

Bayesian Network Node Labels Abbreviation Expansion Mapping	
W:	Day of the Week
H:	Hour of the day
M:	Minutes
CMU:	Current Memory in Use
PMU:	Previous Memory in Use
FMU:	Future (Predicted) Memory to Use
PerFMU:	Future (Predicted) Memory to Use Affected by Periodicity
VarFMU:	Future (Predicted) Memory to Use Affected by Variance
FTM:	Future (Predicted) Total system Memory

Table 5.1: Abbreviation expansion mapping of Bayesian network node labels

5.1.1 Chosen Discrete States for the Nodes

The memory node states were first given in percentages of the total memory of the guest (as explained in section 4.3.3.1). This choice was made to reduce and unify the number and names of the states for all of the guests. As easily seen in the table 5.2, an example is to set the following 14 discrete memory node states for two guests with different total amount of memory. If the states are given in a rate of the total memory, they are the same for both of the guests. If they are given in absolute numbers, they differ when the total memory differs.

Memory Node States Example			
Guest with 1GB of Total Mem		Guest with 2GB of Total Mem	
Rate Node	Absolute Node	Rate Node	Absolute Node
0% - 10%	0MB - 100MB	0% - 10%	0MB - 200MB
10% - 20%	100MB - 200MB	10% - 20%	200MB - 400MB
20% - 30%	200MB - 200MB	20% - 30%	400MB - 600MB
30% - 40%	300MB - 200MB	30% - 40%	600MB - 800MB
40% - 50%	400MB - 200MB	40% - 50%	800MB - 1000MB
50% - 60%	500MB - 200MB	50% - 60%	1000MB - 1200MB
60% - 70%	600MB - 200MB	60% - 70%	1200MB - 1400MB
70% - 80%	700MB - 200MB	70% - 80%	1400MB - 1600MB
80% - 90%	800MB - 200MB	80% - 90%	1600MB - 1800MB
90% - 100%	900MB - 200MB	90% - 100%	1800MB - 2000MB
100% - 110%	1000MB - 1100MB	100% - 110%	2000MB - 2200MB
110% - 120%	1100MB - 1200MB	110% - 120%	2200MB - 2400MB
120% - 130%	1200MB - 1300MB	120% - 130%	2400MB - 2600MB
130% - greater	1300MB - greater	130% - greater	2600MB - greater

Table 5.2: Bayesian memory node states given in discrete percentages versus discrete absolute values. Note that the percentages are more than the total memory of each guest. This is because memory swapping might be observed, that means more than the total available memory needs to be used.

As can be seen from the table 5.1, the first three (W, H, M) of the nodes used are date related nodes affecting the periodicity of the results, and they are always used as evident (known) nodes. The rest are memory related nodes (currently observed memory, previously observed or expected etc) that they might be either evident or query nodes. The discrete states for the memory related nodes were defined as percentages of total memory or actual memory states (see table 5.2) and the quantization of the date nodes was made as the table 5.3 demonstrates.

Date Node States Example			
W (Exact)	W (Boolean)	H	M
Monday	True	00	00 - 04
Tuesday	True	01	05 - 09
Wednesday	True	02	10 - 14
Thursday	True	03	15 - 19
Friday	True	04	20 - 24
Saturday	False	05	25 - 29
Sunday	False	06	30 - 34
		07	35 - 39
		08	40 - 44
		09	45 - 49
		10	50 - 54
		11	55 - 59
		12	
		13	
		14	
		15	
		16	
		17	
		18	
		19	
		20	
		21	
		22	
		23	

Table 5.3: Bayesian date related node states. The days of the week first were given as absolute days (7 states), but then it was converted to a boolean node to increase computing speed and learning efficiency of the Bayesian network

5.1.2 BN Affected Only by Date

The very first Bayesian network evaluated was before the implementation of the dynamic memory allocation. It was based only on periodicity, and composed of 4 nodes (figure 5.1). Three of them are the evident nodes (W, H, M) and one of them is the memory node (FMU) which will give the predictions.

This network performed satisfactory and its results for test server 1 with the highest generated load (look at section 4.1.1) can be seen in the following figures.

Figure 5.2 shows the predictions in red colour and the observed memory in blue colour. The black straight line is the total memory in the system which is not changing because there is no dynamic memory allocation yet in place. The result is quite interesting as the two lines are almost attached to each other.

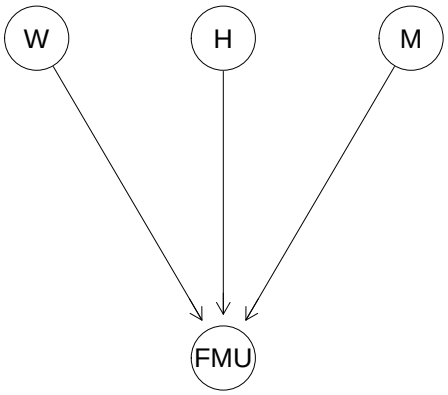


Figure 5.1: Bayesian Network 1 which is affected only by date. W, H, M nodes are the evident nodes and FMU is the query node which will give the prediction given in a percentage of the total memory. Expansion of the abbreviations for each node in table 5.1

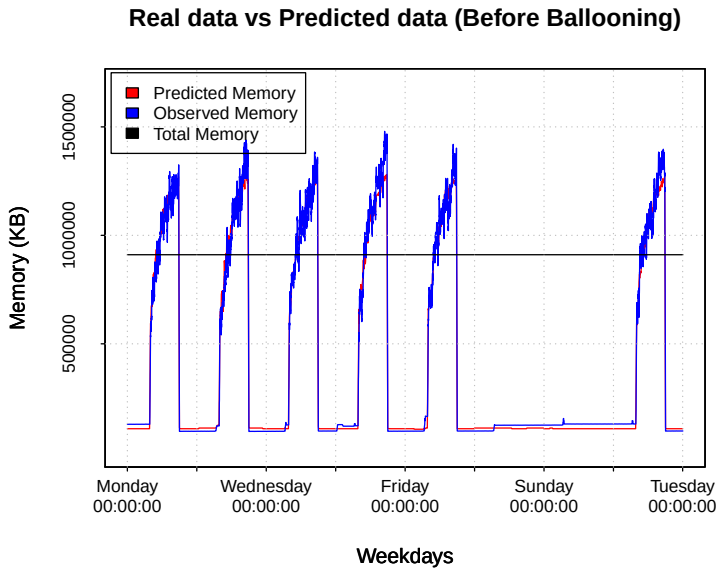


Figure 5.2: Prediction affected only by date before ballooning. Real data (blue line) plotted versus predicted data (red line)

Figure 5.3 illustrates exactly the same as figure 5.2, but this time +15% more memory (explained in section 4.4.1.2) is added to the prediction. Now the prediction is clearly a little bit over the actual memory observed.

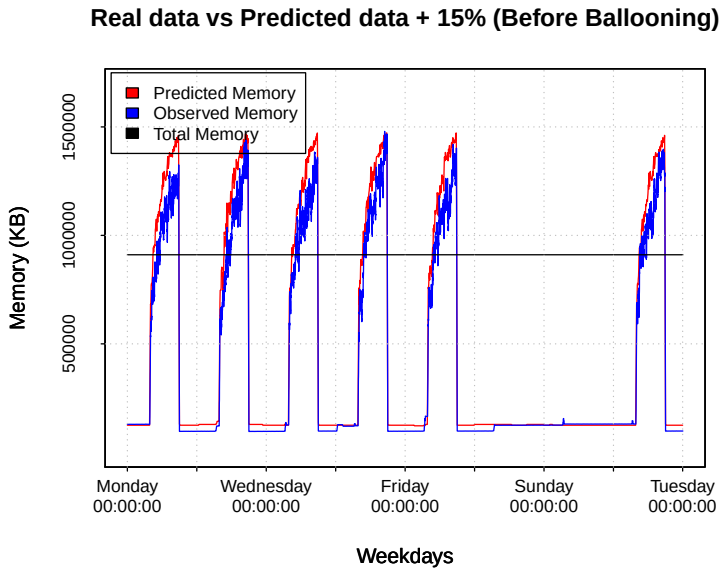


Figure 5.3: Prediction affected only by date before ballooning. Real data (blue line) plotted versus predicted data (red line). 15% is added to the predicted data which puts it above the observations for most of the time.

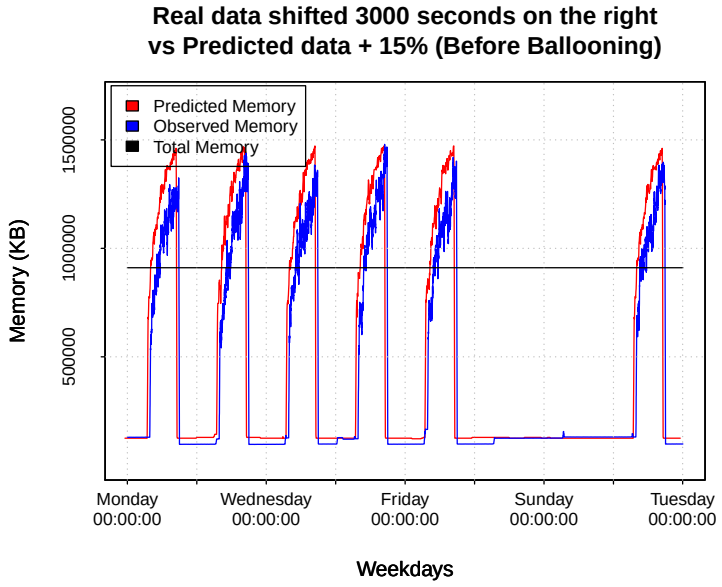


Figure 5.4: Prediction affected only by date before ballooning. Predicted data comes first.

Figure 5.4 is plotted only for demonstration reasons and it shows that the prediction comes first and then the observed data. Because this is a plot for a whole week, the prediction has been shifted for 3000 seconds to be visible in

the graph. The actual predictions are very short term predictions coming only 300 seconds before.

Figure 5.5 shows the absolute difference of the predicted (without the +15%) versus the observed data, and the boxplot in figure 5.6 explains it in actual numbers. The median value observed is 15.88MB far from the predicted one, which means these two graphs are very close. Even though some values differ as much as 250MB, they are not so many and they are classified in the outlier range (check section 3.5.1). They form the 18.76% of the total predictions with an average value of 68.26MB which is again pretty close.

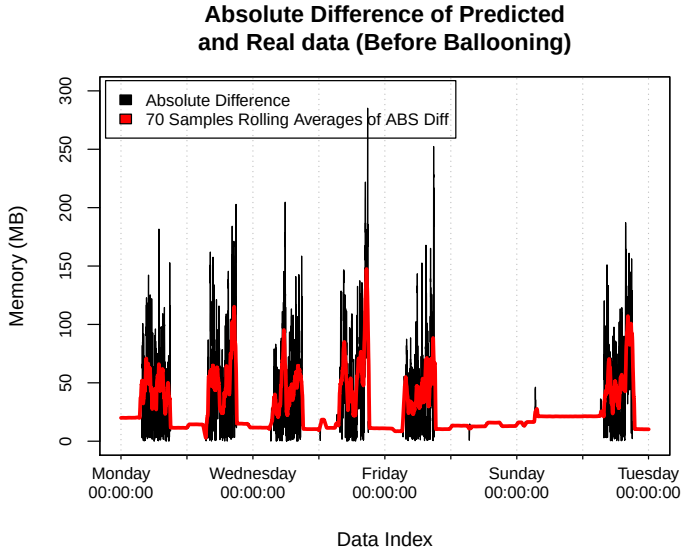


Figure 5.5: Prediction affected only by date before ballooning. Absolute difference of predicted and observed data. The rolling average line has been plotted to give a better viewable graph when comparing the numbers with the boxplot in figure 5.6.

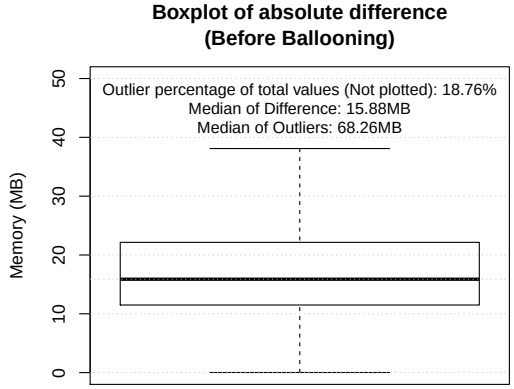


Figure 5.6: Prediction affected only by date before ballooning. Boxplot of the absolute difference of predicted and observed data.

5.1.3 BN Affected by Date and Total Memory

In section 5.1.2 the Bayesian network used a memory node given in percentages of the total memory. When the dynamic memory allocation was in place and ready to operate, a challenge appeared for the initial network. The maximum total memory would start changing, so one more node (FTM) with actual memory states representing the total memory at each time of the day introduced as depicted in figure 5.7. This change was made to demonstrate also the modularity of the Bayesian networks (In the previous network added one more node to make it fit the current conditions and give extra information). Then the software would first question the network for the expected total memory, and then for the predicted memory which is given in percentages of the first questioned node.

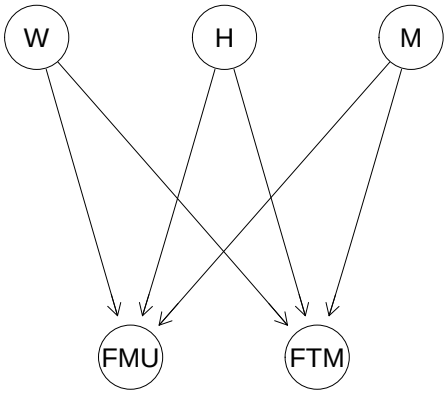


Figure 5.7: Bayesian Network 2 which is similar to 5.1. Another node to predict the total memory as well has been added.

Figure 5.8 shows that since ballooning started, the behaviour of the whole system changed unexpectedly. The observed data now goes surprisingly higher than what it was observed so far without the dynamic memory allocation in place. In figure 5.2, the blue line is not reaching values more than 1400000KB of memory, while in figure 5.8 the blue line almost touches 2000000KB of used memory at certain points. This happens because in the first case without the ballooning, the data generation proved to be too demanding for the capacity of this host and the host was heavily swapping. A consequence of this, is a heavy slowdown and limitations for the running processes. When the dynamic memory allocation started, the host was set "free" to use more memory that it was needed to satisfy the needs of the system, thus why the observed used memory increased a lot. Something else important to be noticed here is that the prediction (red line) is not so good as it is far below the observed data in some points of the graph and it looks pretty similar with the predictions

made in figure 5.2. This is a natural outcome since the system has learned from the data collected so far, and the evident nodes are related only to periodic parameters (the date). The results are similar to these of a system giving the mean value from observations in a specific time of the day. Even though the system made some new observations now, these would take some time to have a significant effect in the results of the predictions, because the learning dataset is more than 30 days long, so 1 day of different observations would not make an instant difference.

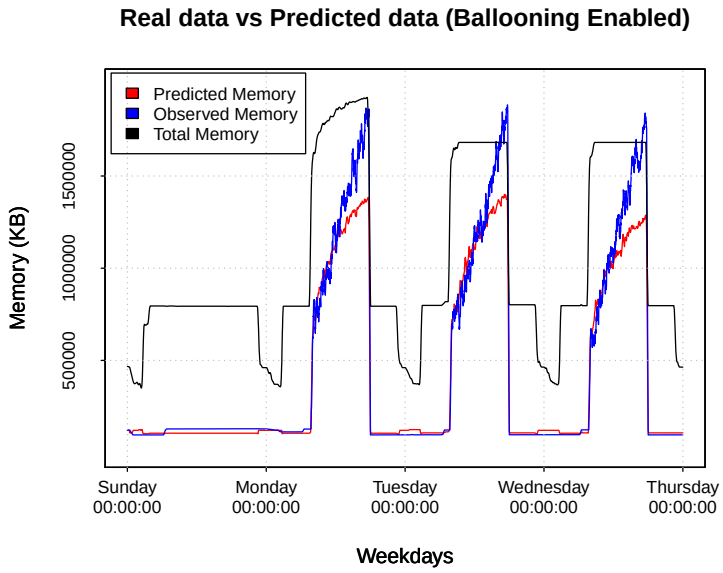


Figure 5.8: Prediction affected by date and total memory after ballooning started. The ballooning is obvious as the total memory of guest (black line) is constantly changing. The prediction now (red line) is not as good as in the first Bayesian network, and the observed memory has increased a lot without any change in the processes running in the system.

The boxplot in figure 5.9 shows that the median value of the absolute difference of the observed versus the predicted data is very low (22.9MB), but one can tell that there are some very unsuccessful predictions by looking at the whiskers (the top one extends up to 105MB) and the median value of the outliers which is 242.7MB and represents the 16.61% percent of the predictions. The missed predictions reach values up to 600MB as can be seen in figure 5.10.

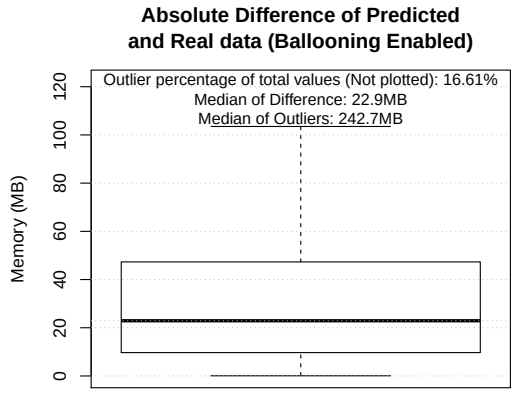


Figure 5.9: Prediction affected by date and total memory after ballooning started. Boxplot of the absolute difference of predicted and observed data.

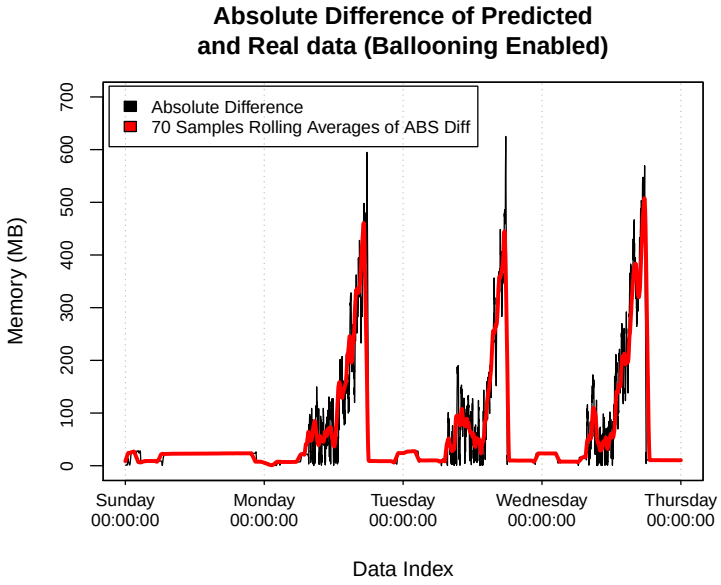
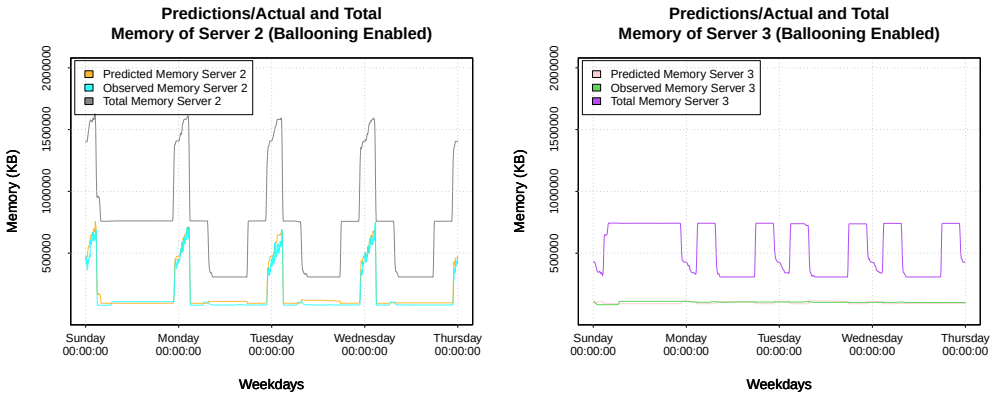


Figure 5.10: Prediction affected by date and total memory after ballooning started. Absolute difference of predicted and observed data. The rolling average line has been plotted to give a better viewable graph when comparing the numbers with the boxplot in figure 5.9.

The last noticeable thing from the graph in figure 5.8, is the continuous change of the total memory (black line). This is normal since the ballooning is enabled in this case and it is driven by the predictions, although the given memory is more than the predictions and this is caused by the added memory (+15%) and the extra added memory described in section 4.4.1.4. Every night a little after midnight, there is also a dent in the total memory without a significant change in the predictions before and after the dent. This is caused by Bayllocator because at this specific time, the second test virtual machine test server 2 is

working and it is assigned more memory as illustrated in figure 5.11a. A similar dent is observed for test server 3 in figure 5.11b, and the summary of these two dents matches the size of the memory lift for test server 2. Similar dents are observed for test server 2 and test server 3 when test server 1 increases its activity from 08:00 to 18:00.

Figures 5.8, 5.11a and 5.11b are all plotted in a single graph for comparison in figure 5.12.



(a) Test Server 2. **(b)** Test Server 3.

Figure 5.11: Prediction affected by date and total memory after ballooning started.

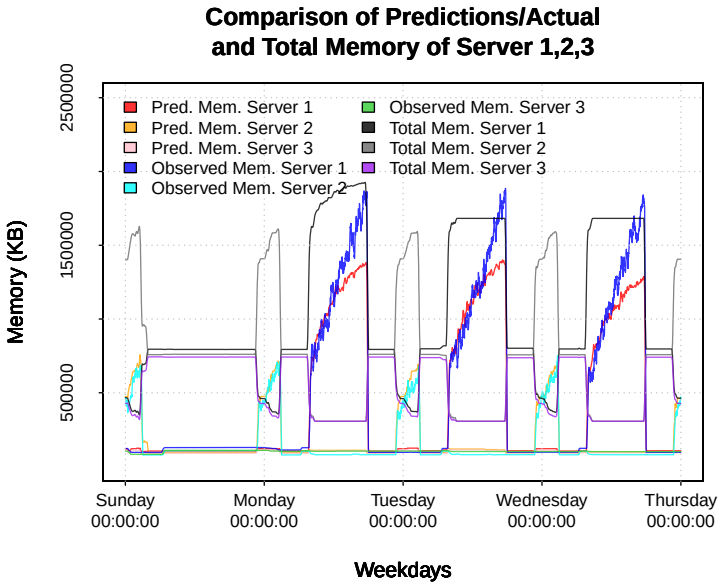


Figure 5.12: Prediction affected by date and total memory after ballooning started. Figures 5.8, 5.11a and 5.11b plotted together for comparison.

5.1.4 BN Affected by Date and Current Memory

The two previously tested Bayesian networks were affected only by periodic factors. In a sudden change of memory or new observation, periodic only prediction is not effective as proved in the previous section. The reason is that this kind of network is almost immune to new observations if the learning dataset is big enough, because it acts in a similar manner to a system calculating the means of the total observations (a new value in a large dataset will not affect the mean value significantly). In an effort to surpass this problem, a new evident node was added as probe giving the currently used memory to the Bayesian network. This would hopefully deal with the problem as the system would now be affected by a non periodic factor, which is the current memory in use. If for example it is Monday midnight and there is a sudden increase of the current memory (unexpected event) that it has never been observed before (or it has been observed only once), the system will give a different prediction since the periodic behaviour will not be followed any more.

5.1.4.1 Current Memory Given in Percentage

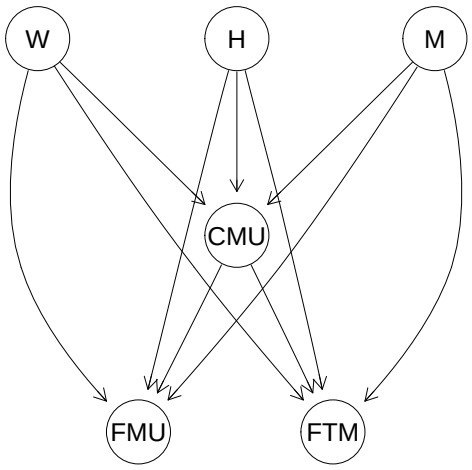


Figure 5.13: Bayesian Network 3. Similar to Bayesian network 2 in figure 5.7 with a new node added (CMU) to break the periodic only predictions.

The Bayesian network in figure 5.13 was created, which is a potential improvement over the previous Bayesian network test in figure 5.7. A new evident (known) node has been added (CMU) which is affected by the W, H and M nodes but this does not really matter as this node will always get a known value which is the currently used memory of the guest system. This node in turn will affect the predicted total memory (FMT) and the predicted memory percentage of the total memory (FMU). The training dataset for the node FMU

and CMU is the same, with the latter one shifted some minutes backwards because the current memory comes obviously before the predicted memory. The prediction in the experiments is 5 minutes in the future, so the CMU training dataset is shifted 5 minutes backwards.

The results of this Bayesian network were very unexpected but explainable and promising. As figure 5.14 demonstrates, there is a very unstable behaviour even for the periodic events (between 08:00 and 18:00). This is related to the memory states for CMU and FMU given in a percentage of total memory. CMU is given in a percentage of the current total memory in the system, while FMU is given in a percentage of the predicted total memory FMT. Because ballooning is enabled, the current total memory changes so the CMU changes a lot and because it takes unobserved values as input, it gives this bouncing prediction which leads to dynamic memory allocation bouncing for all of the virtual machines, since Bayllocator will adjust the memory to all of them when it is running (similar to what was explained in figure 5.12). An unobserved value means that this combination of evidence has never been met before and it does not exist in the conditional probability table of this Bayesian network. The predictions following an imaginary straight line at around 900000KB of memory are unknown answers (in the following section 5.1.4.2 there is an explanation why the unknown answers get a fixed value which in this case is around 900000KB).

To clarify the cause of this behaviour, consider the following example: test server 1 is using 600MB of memory, it has a total of 1200MB, and CMU takes a value of 0.5 (600MB/1200MB). The Bayesian network has learned that when the CMU takes a value of 0.5 say on Monday at 15:00, it has to assign 1200MB (the same) of memory. Adding also the +15% and the extra memory as described in section 4.4.1.4, Bayllocator assigns 1600MB. In the next run, the server is using 700MB of memory but now the total memory has been set to 1600MB from the previous prediction and dynamic allocation based on this. The CMU instead of now taking a greater value than before, it will take a value of 0.4375 (700/1600) which is unexpected and it gives a prediction around 900000KB of memory which it might be 1200MB after adding the extra memory. In the next run the CMU might get again 700/1200=0.583 which is an observed value for this day and this hour so it will give a valid prediction and so on.

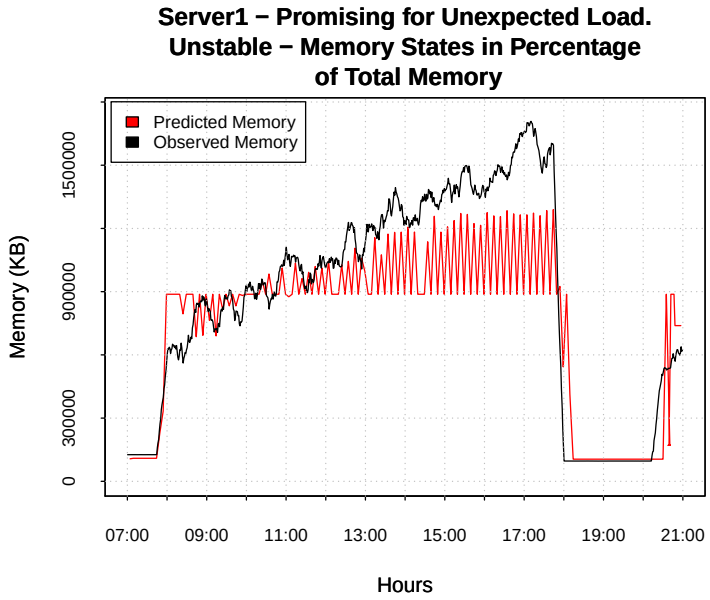
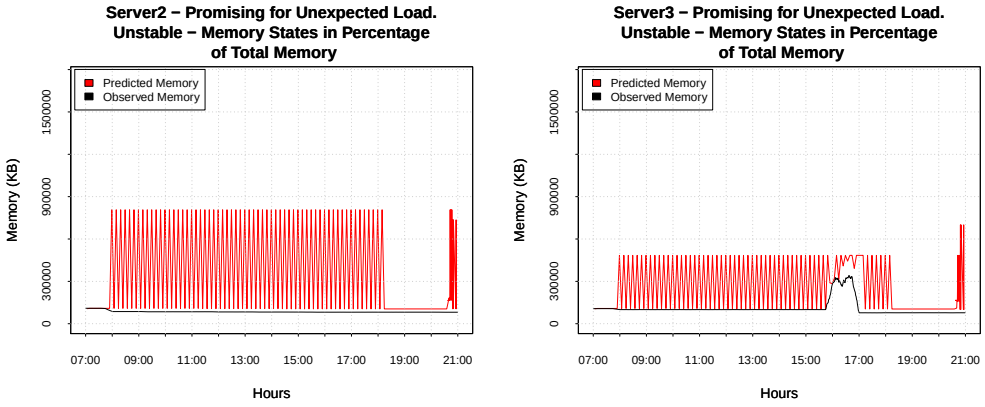


Figure 5.14: Prediction affected by date and current memory - Unstable behaviour but a promising result is seen between 20:00 and 21:00 in an unexpected memory load.

The promising part of this Bayesian network lies in the prediction between 20:00 and 21:00. Some memory load is noticed that it has never been observed before, but this Bayesian network reacts in this unexpected memory load and gives an unknown answer instead of the standard periodic answer which would not be affected at all.

The figures in 5.15a and 5.15b shows that due to the dynamic memory allocation, whenever a change is made to one of the virtual machines (Test Server 1 in this case), all of the predictions are affected since the CMU is related to their current total memory and this is changing.



(a) Test Server 2 unstable behaviour. **(b)** Test Server 3 unstable behaviour.

Figure 5.15: Prediction affected by date and current memory

5.1.4.2 Current Memory Given in Actual Values

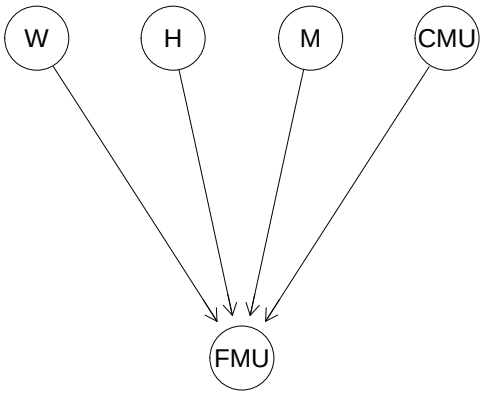


Figure 5.16: Bayesian Network 4. FMT node has been removed and the CMU and FMU node states use the actual memory values and not the percentage of total memory any more.

When the memory states are given in a percentage of total memory, prediction instability is introduced with the current design as explained in the previous section 5.1.4.1. To resolve this weakness, the previous Bayesian network in figure 5.13 was changed and the memory states are not given in percentages of total memory any more. As a result of this change, the total memory node (FMT) is not needed and the simplified tested network looks like the one in figure 5.16 with four evident and one query nodes.

The resulting predictions of this network can be seen in figure 5.17. In this graph from Monday to Tuesday only the normal typical workload exists, while from Tuesday to Wednesday extra memory (unobserved) variation is introduced from "Start Var" to "Stop Var" marks in x axis (a second instance of loadsim.pl is running), and it keeps on running even when the typical daily workload ("T.W. Starts" to "T.W. Stops" marks in x axis) is in place.

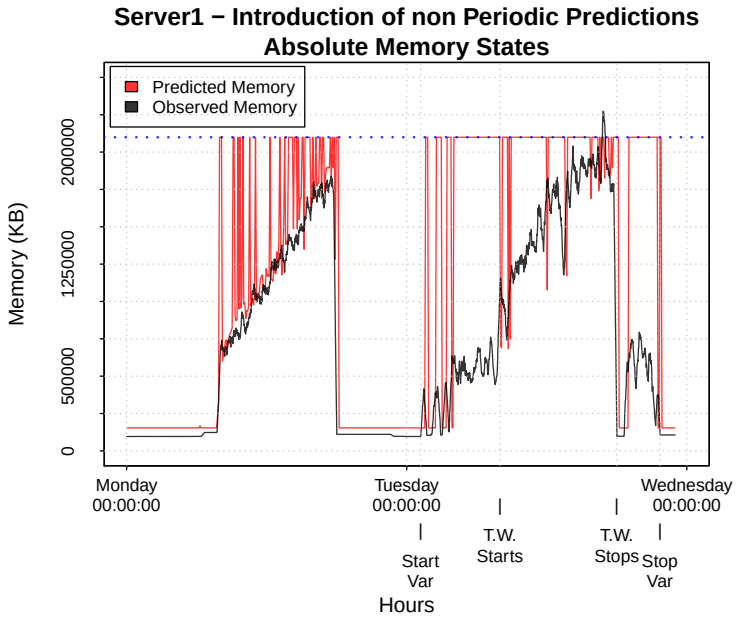


Figure 5.17: The "Start Var" and "Stop Var" marks in x axis, shows when extra random data was generated to study the predictions of non-periodic behaviour. "T.W. Starts" and "T.W. Stops" stands for Typical Workload and these marks shows that from Tuesday to Wednesday extra generated memory load was in place on top of the typical workload. The blue dotted lines indicates the value of unknown answers given by the Bayesian network.

From Monday to Tuesday the predictions succeed to follow the workload until the peak load, in contrast to figure 5.8 which was only affected by periodic factors. Even if it succeeds in one part it fails frequently, as many of the predictions in this Bayesian network often reach a certain wrong value in y axis, marked with a horizontal dotted blue line. This value is exactly 2099200 KB and it is an indication that the system does not know how to answer for the given evidence. When the Bayesian network gets an unobserved combination of evidence, then all of the probabilities of the query node are equal. Because the discrete states defined for the memory nodes CMU and FMU are 41 (from mem_0_100, mem_100_200 to mem_4000_greater every 100MB), the probability for each state is 1/41 = 0.024390244 which gives this number as expanded in equation 5.1.

$$\sum_{n=100,4000} n \cdot \frac{1}{41} + \sum_{n=1}^{39} (n \cdot 100 + 50) \cdot \frac{1}{41} = 2050MB \tag{5.1}$$
$$2050MB \cdot 1024 = 2099200KB$$

From Tuesday to Wednesday most of the predictions hit this value because this extra generated data on top of the typical workload was introduced for the very fist time so it is a brand new observation for the system. These wrong predictions can be fixed either with a larger training dataset (more observations),

or a more intelligent Bayesian network design which will prevent unobserved combinations of the evident nodes as much as possible.

If figure 5.18 the missed predictions presented in figure 5.17 have been pseudo fixed, by choosing the previously predicted value for each prediction with the exact value of 2099200. This graph shows that the rest of the predictions are better than in any of the previous networks tried so far.

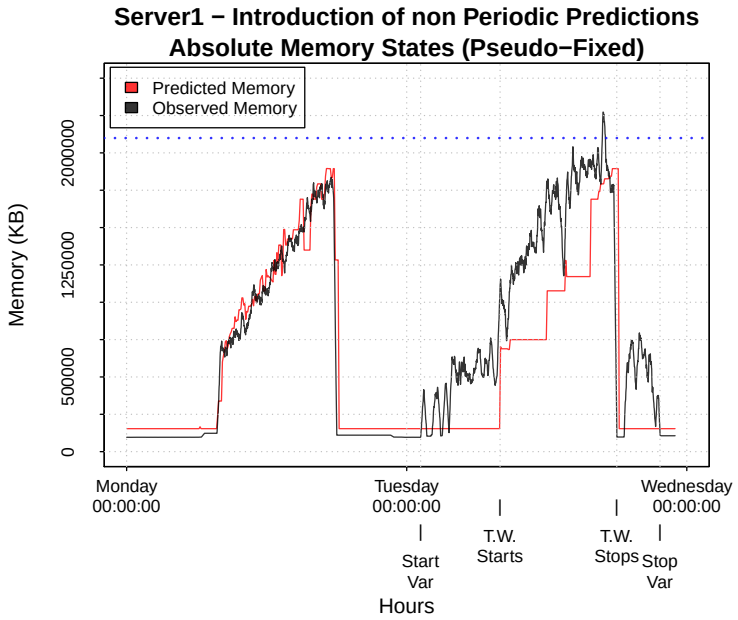


Figure 5.18: Same as graph in figure 5.17, but the missed answers (2099200 KB) have been pseudo fixed to promote the known answers by using the previously known prediction.

5.1.5 BN Affected by Current Memory and Previous Memory

The Bayesian network in section 5.1.4.2 gave many unknown predictions, and the assumption was that they are related to the lack of observations. To see if the assumption is valid, the simple network in figure 5.19 was created. It is not related to the date but only currently observed memory (CMU) and observed memory from the previous run (PMU).

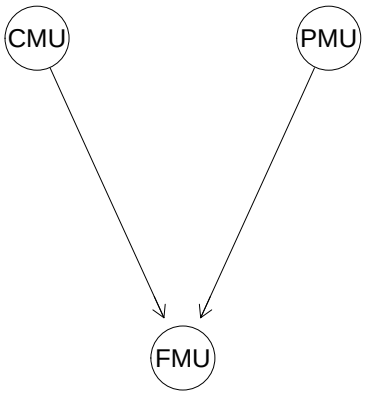


Figure 5.19: Bayesian Network 5. Affected only current memory (CMU) and previously observed memory (PMU).

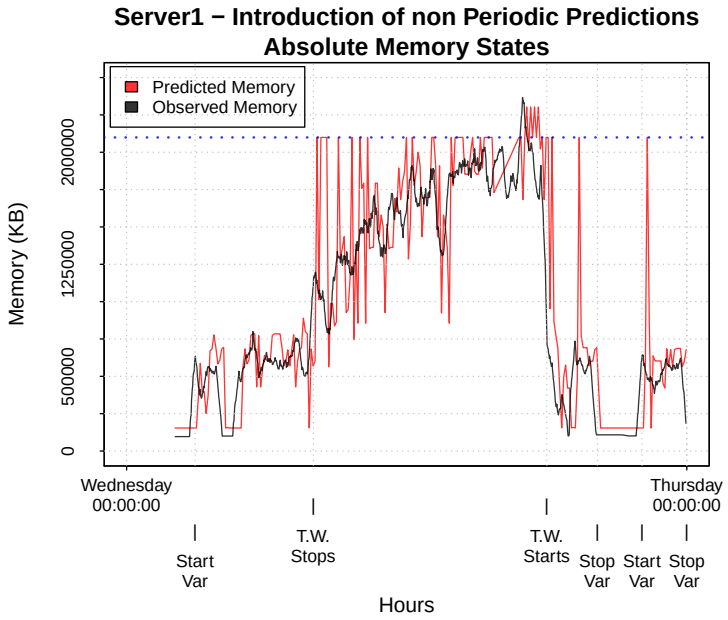


Figure 5.20: Much less unknown predictions without any date related nodes.

The results in figure 5.20 confirmed that date related nodes make the learning procedure slow, as much less unknown predictions occurred in this test (compare figure 5.20 with 5.17). The overall result is not very good for proactivity, but it could work well for reactivity if the observed data was not totally random (due to the way the data is generated), which it is usually not in real life server load.

5.1.6 Various Combinations

After all of the previous Bayesian networks tried it was concluded that date nodes can give answers related to periodicity, but they decrease the flexibility of the system to learn fast. The memory related nodes can learn much faster, but they do not offer proactivity since the memory itself which the prediction is made for is involved. A combination of both would be ideal and some more networks tested out without significant success.

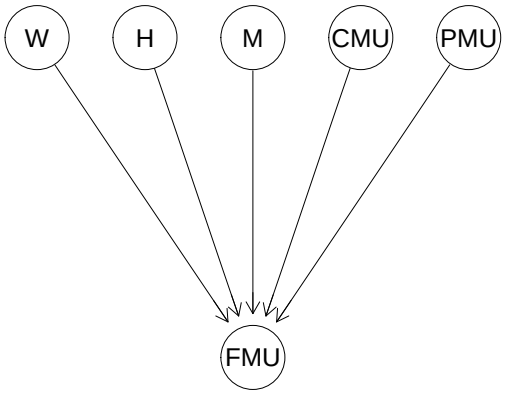


Figure 5.21: Bayesian Network 6. This one could not be tried due to a memory leak of gRain.

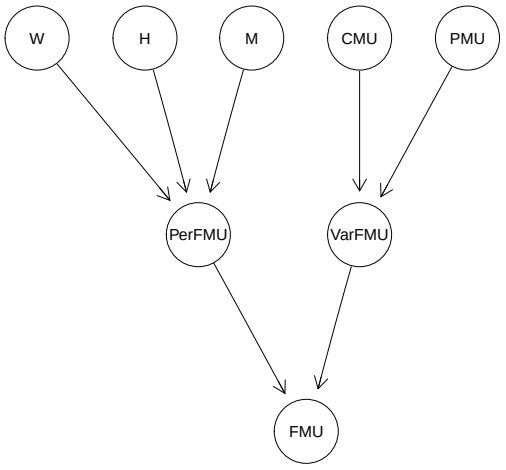


Figure 5.22: Bayesian Network 7. This network did not give any better results from some of the previous tested networks.

The Bayesian network in figure 5.21 could not be run at all (gRain was memory leaking with this network in a system with 6GB of RAM), while the one in figure 5.22 was an idea to separate the final prediction based on a prediction affected by periodic factor (PerFMU) and a prediction affected by memory variations (VarFMU) which did not succeed.

5.1.7 Final BN - Affected by Date and Current Memory

Finally, it was chosen to use the Bayesian network described in section 5.1.4.2 with a small change to make it more flexible to learn from new observations. Instead of using the exact day, the "W" node was converted to a boolean (see table 5.3) which takes a true value from Monday to Friday and a false value for the weekend. Now if a new event is observed on a Monday at 20:00, it will be unknown. If the same event occurs on Tuesday at 20:00 it will not be unknown (as it would be with the exact days for node "W") because the "W" node has the same value for both Monday and Tuesday and the rest of the evident nodes are the same (H=20, M=00-04, CMU=Related To The Event). Figure 5.23 (and the pseudo fixed 5.24) illustrates that now the system can make some correct and well targeted predictions even for the non periodic memory load outside the typical workload borders.

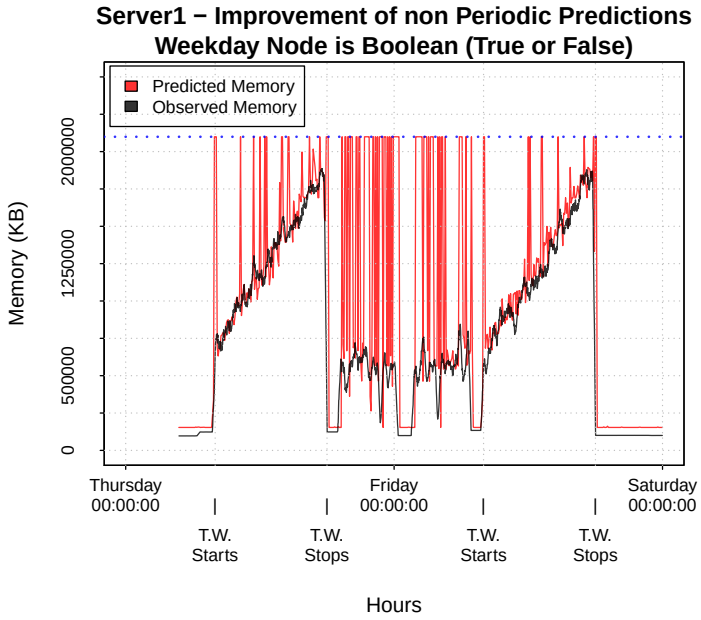


Figure 5.23: Results of Bayesian network in figure 5.16, with "W" node changed to boolean states.

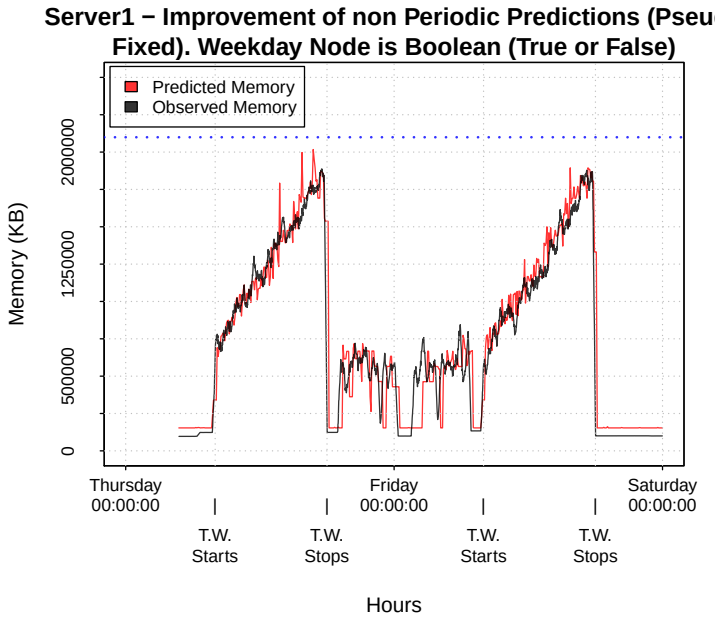


Figure 5.24: Pseudo fixed graph (unknown predictions equal to the previous known predictions) of figure 5.23, shows some well targeted predictions.

One last change was made which improved the predictions of the eventually used network a lot. This was the training data set and the smoothed data the system was fed with. All of the previous graphs and predictions were made using 15 minutes rolling averages, while the prediction was 5 minutes short term prediction and the CMU was taking as evidence the average of the last 5 minutes. This means that the training dataset was quite different because the 15 minute average is much more smoothed than the 5 minutes average, so more unobserved states appeared at given times. A comparison of the typical workload between figure 5.23 and 5.25 instantly shows this improvement. Also if one takes a careful look at the black lines (observed memory), the one in figure 5.25 has much more variation due to the shorter averages.

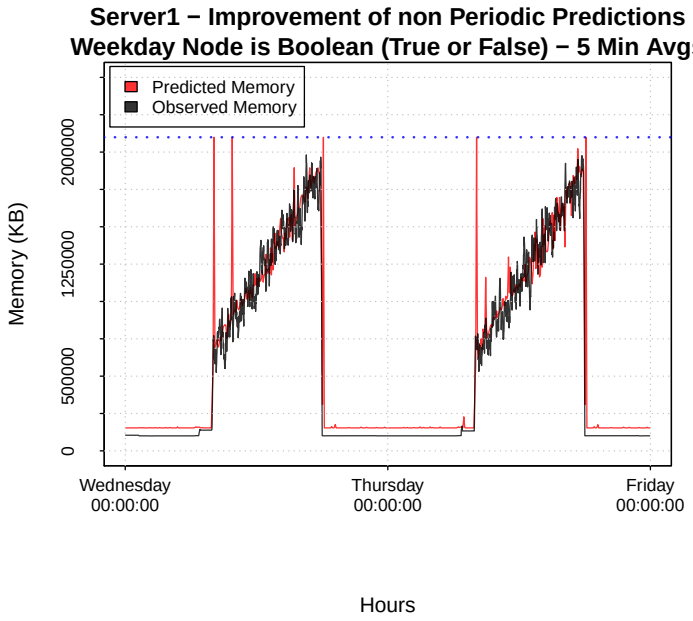


Figure 5.25: Changed the training set to 5 minutes rolling averages improves the results a lot.

5.1.8 Application to Real Data

After the extensive tests in the test environment, a simulation applied against collected data from real servers. The data collection period was almost one month, and three different training datasets were used to make predictions. The common predictions for all of the simulations per host are after the vertical dotted black line in the graphs to follow.

5.1.8.1 studssh.iu.hio.no

As figure 5.26 demonstrates, studssh has a very steady and linear increase of its memory load. The first learning dataset is from April, 4th 2012 to April 16th 2012 but it fails to predict correctly the days right after it because the system still observes new events (the memory keeps on increasing). Near the April, 24th there is a sudden memory drop to observed states included in the learning dataset, and the predictions are good. Around April, 27th the system was rebooted and from this moment after, all of the observations are new (prediction always at 2099200 KB).

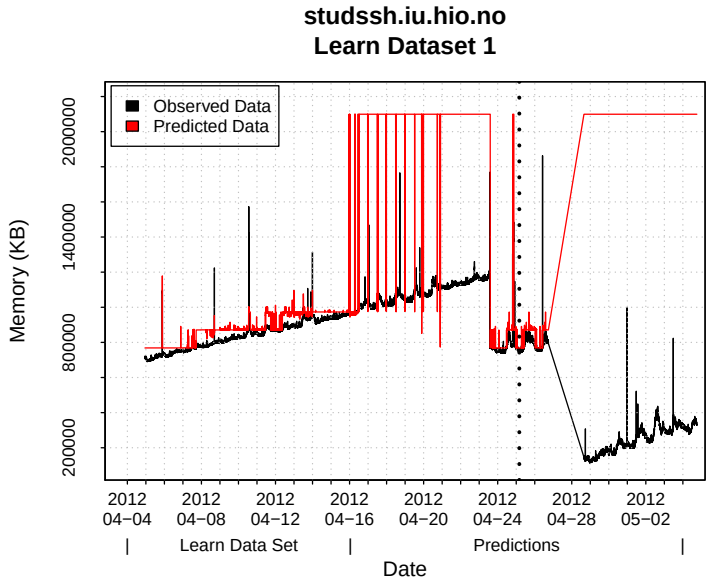
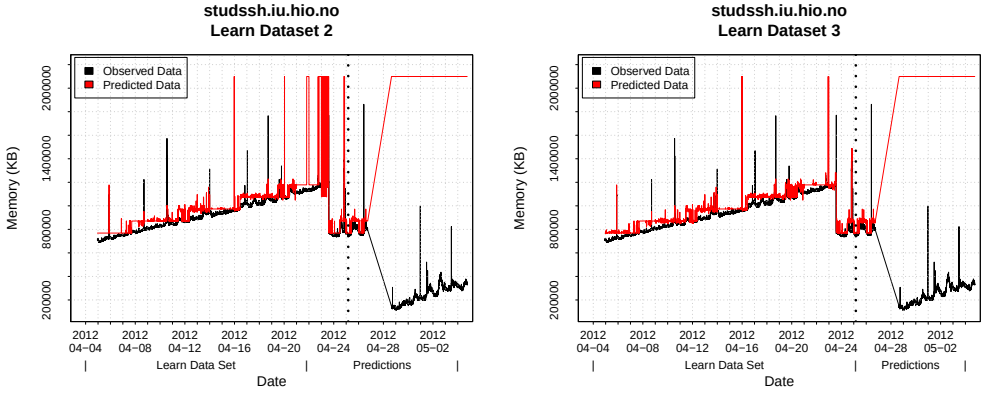


Figure 5.26: studssh.iu.hio.no - Learning Dataset from April, 4th 2012 to April 16th 2012. Predictions right after the learning dataset till the end of the graph.

Learning dataset 2 was from April, 4th 2012 to April 22nd and learning dataset 3 from April, 4th 2012 to April 25th as can be seen in figures 5.27a and 5.27b. Nothing very interesting to notice as there is no periodicity observed here.



(a) Learning Dataset from April, 4th 2012 to April 22nd 2012. Predictions right after the learning dataset till the end of the graph.

(b) Learning Dataset from April, 4th 2012 to April 25th 2012. Predictions right after the learning dataset till the end of the graph.

Figure 5.27: Real Data Analysis - studssh.iu.hio.no

5.1.8.2 nexus2.iu.hio.no

Nexus2 gave some more interesting results as it shows some periodicity. The first learning dataset for nexus2 is from April, 10th to April 16th (figure 5.28), the second from April, 10th to April, 22nd (figure 5.29a), and the third one from April, 10th to April 25th (figure 5.29b). All of the three simulations ran have missed predictions, but it is obvious that when the system is trained with a larger dataset it fails much less. A comparison of the common predictions for all of the simulations (after the vertical black dotted line) shows that for the first training set, 808/2774 predictions failed (29.1%), for the second 122/2774 (4.4%) while for the third only 76/2774 which makes up the 2.7% of the total predictions.

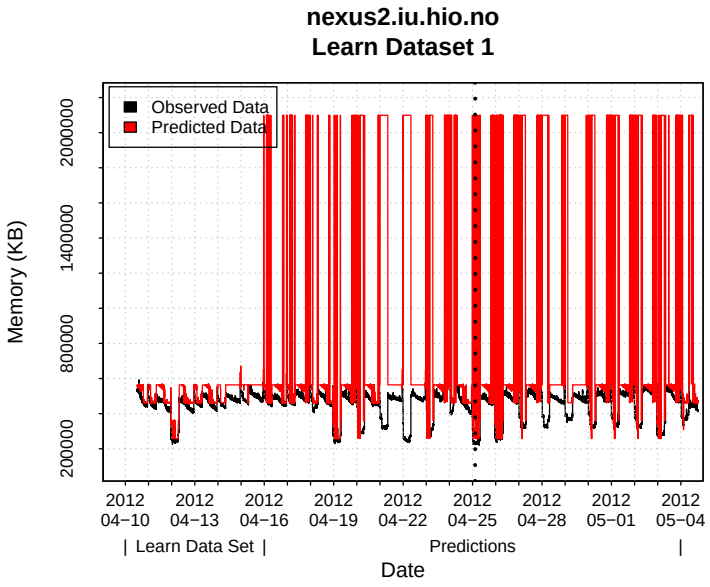
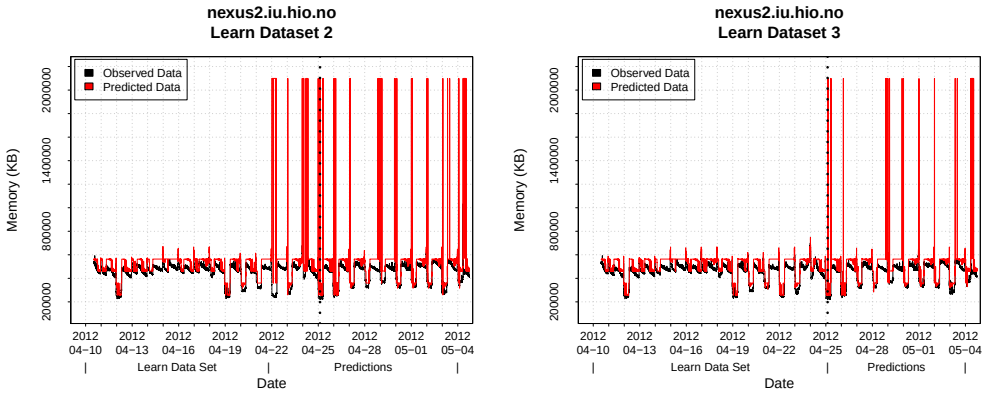


Figure 5.28: Learning Dataset from April, 10th 2012 to April 16th 2012. Predictions right after the learning dataset till the end of the graph.

If figure 5.30 the absolute difference of the predictions versus the observed data is illustrated for the period after the dotted lines in figure 5.29b. The 76 unknown predictions have been removed. The boxplot in figure 5.31 shows that the median value of the difference is 57.37MB and the median of the outliers is 168.78 forming only the 0.35% of the predictions.



(a) Learning Dataset from April, 10th 2012 to April 22nd 2012. Predictions right after the learning dataset till the end of the graph.

(b) Learning Dataset from April, 10th 2012 to April 25th 2012. Predictions right after the learning dataset till the end of the graph.

Figure 5.29: Real Data Analysis - nexus2.iu.hio.no

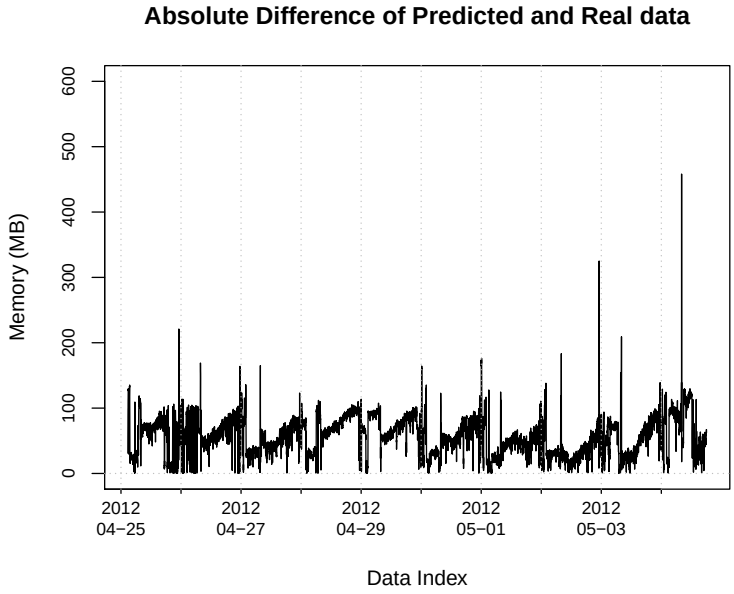


Figure 5.30: nexus2.iu.hio.no - Absolute difference of predicted versus observed data.

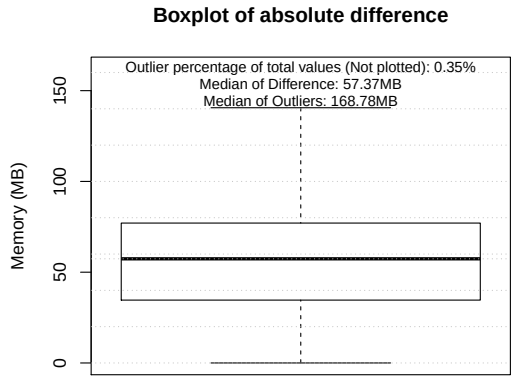


Figure 5.31: nexus2.iu.hio.no - Boxplot of absolute difference of predicted versus observed data.

If one takes a better look to the predictions both of studssh and nexus2, they will notice that the predictions follow steps. This is because the training datasets are based on discretized and not continuous variables, so as the predictions. If the memory in the system is 501MB or 600MB, both of these observations are encoded as the memory state mem_500_600 and they represent a memory state of 550MB in the reverse operation.

5.2 Bayllocator Evaluation

A simple evaluation was made to see the positive effect on performance when using dynamic memory allocation driven by Bayllocator. Two operations were running in test server 1 and their execution time was measured as described in section 3.6.1. As can be seen in figure 5.32, the execution time of a PHP script running on an apache 2 web server would take as much as 50 seconds under heavy memory load (between 08:00-18:00), while it would not take more than 3 seconds under normal circumstances. When ballooning initiated, the operation looked like normal at any time. Same observations can be made for figure 5.33 where a secure copy operation was initiated from an external machine to test server 1. The reason for this slowdown before ballooning, is the heavy memory swapping on hard disk. Hard disk is several orders of magnitude slower than memory. To avoid this kind of slowdowns and use the most out of the physical hypervisors, the dynamic memory allocation is necessary.

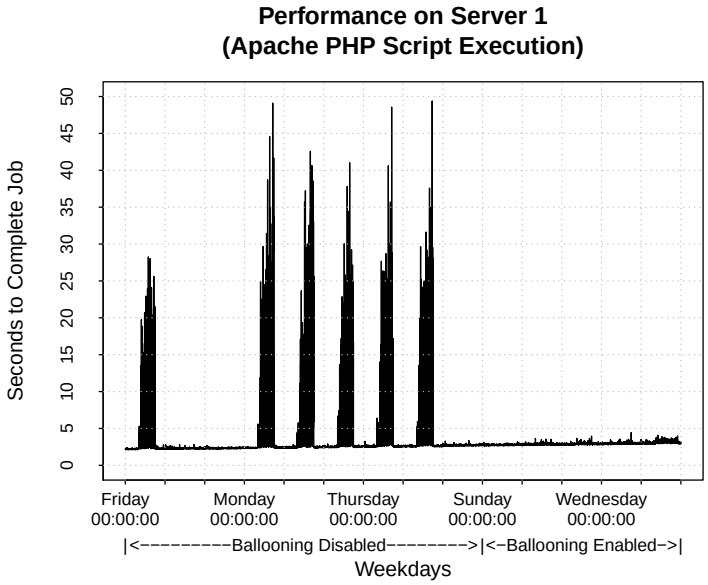


Figure 5.32: Performance of PHP scrip execution

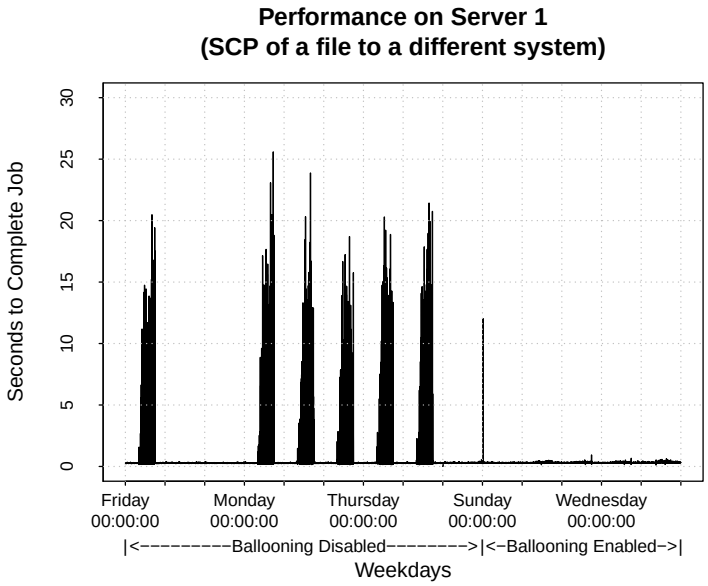


Figure 5.33: Performance of secure copy operation

Chapter 6

Discussion and Future Work

This chapter discusses the different parts of the implemented solution, and suggests improvements for future work and continuation of this project.

6.1 Evaluation of the Project as a Whole

The competence of Bayllocator resides in the prediction method to predict system utilisation in a non traditional way, using Bayesian networks. The focus is mainly in memory prediction and appliance of dynamic memory allocation based on this information to improve sharing efficiency. Due to its nature, memory is not as easy to share as time shared resources like CPU or Disk I/O, becoming an obstacle to server consolidation, green and low cost ICT. Highly active ongoing research tries to predict system utilisation and use this information for dynamic resource allocation, but most of them use traditional statistical methods to approach the problem. The traditional statistical methods might involve rigorous mathematics and they usually are single purpose, or hard to adapt to different needs. Bayesian networks are strongly associated with artificial intelligence, and this fact made it an exciting topic to get involved with. Moreover, after an extensive literature survey made for the background chapter, it does not seem that effort to apply the use of Bayesian networks on server utilisation prediction exists. Bayesian networks differs significantly from the traditional statistical methods, and their big advantage is that they are modular directed acyclic graphs, composed of nodes representing variables related to the system they work with, and changing of these nodes might give totally different results and use cases. Furthermore, even the output from traditional methods can be used as the input in a Bayesian network, and more than one of them can be combined. The modular design gives flexibility and allows the easy combination and alteration of different parameters affecting the system. This was proven in this short term thesis where 7 different Bayesian networks were tested out.

The prototype software made in Perl, which is calling an R script for the Bayesian

predictions. This modular design (standalone predictions) gives the flexibility to change the prediction method at any time, and evaluate the guest performance under different circumstances. The results are satisfactory but only the beginning, as there is plenty of space for improvements, benchmarks and comparisons of different ballooning techniques under real server load.

The biggest challenge throughout the accomplishment of the project was time constraints, and lack of wide purpose collected data from real servers to allow further experimentation with the causal nodes affecting the final memory predictions. Each change to the Bayesian network needed time to be tested and make assumptions on how well it works. But when one gets used to work with Bayesian networks, there is a broad spectrum of applications this method can be applied to, and certainly computing utilisation prediction and decision making is one of them.

6.2 Data

This section discusses the generated and collected data.

6.2.1 Generated

One of the problems in the beginning of the project, was the lack of collected data which should be generated for the experiments as explained in sections 3.3 and 4.1.1. Data was collected and stored in a database every one minute for each of the three running guests, resulting in a big table. Because of the data randomness, rolling averages decided to be created in the database, but this proved to be challenging because of the large growing table[1]. No operation tried internally in MySQL was fast enough to be acceptable (MySQL Views, Calculations), so a separate Perl script was created for this purpose and it was set to update an intermediate table with the averages once per day.

One surprising result at a glance, was the increased used memory on test server 1 when ballooning driven by Bayllocator was introduced (compare figure 5.2 before ballooning and 5.8 after ballooning initiated). This behaviour is attributed to memory swapping on hard disk and application slowdown because of this. All of the memory usage related graphs presented in this thesis, use memory and swap in use (without memory caching and swap caching) as the actual total memory used by the system.

One more unexplained result, is why the memory load keeps increasing as long the data generation script runs in test server 1. As explained in section 4.1.1, it is set to use maximum of 1000MB with 3 threads, and this would result in an average usage of 500MB (section 3.3 explains more). If someone takes a

[1] http://stackoverflow.com/questions/9746386/mysql-data-smoothing

look in figure 5.25 (and more of the previous figures), when the daily data generation starts the system uses around 600000KB-750000KB of memory which is normal (an average of 500MB including the memory consumed by the operating system and other applications running). However, there is an increment with a stable slope until the data generation stops. This was not investigated much, as it does not affect the scope of the final results except for the the fact that more memory would have to be allocated for this virtual machine.

6.2.2 Collected from Real Servers

Much data was collected from studssh.iu.hio.no and nexus2.iu.hio.no, but there was not enough time to investigate possible correlations of events affecting the memory usage. Also the memory in these two servers do not change dramatically fast to be able to search for an obvious cause that it could be converted to a Bayesian node affecting the prediction. The simulation was then run on the Bayesian network tested with the generated data. Number of logged in users and number of processes consuming memory in the system are two possible parameters which could have been useful, but they were not included in the initial data collection. They were eventually added to the collection script but time constraints prevented this test.

6.3 Bayesian Networks

SamIam, a free and easy to use program with graphical user interface was used to get familiar with Bayesian networks. SamIam supports the design of Bayesian networks, editing of the conditional probability tables of the nodes, EM (Expectation-Maximization) learning and querying [76]. It includes also a command line utility to make queries to Bayesian networks which was considered to be used in the beginning of the project, but the lack of complete command line functionality (Bayesian network design, training and querying), would not allow the fully automated establishment of the setup. R statistics language was used instead.

6.3.1 Limitations

Discrete Bayesian networks were evaluated which is the most common type of Bayesian networks, and some limitations were observed with continuity in data. A closer and careful look at the predictions (especially those from the simulation on real servers in figures 5.26 and 5.28), illustrates that the predictions follow steps. This is because the training datasets are based on discretized and not continuous values. If very high accuracy is needed, a single state for each value should be used, but sometimes this might not be possible. In this experiment one state for each 100 MB was used, and 41 of such states

for a single node are needed for guests with 4GB of memory. If one state for each 1 MB was needed, 4000 states for one node would be needed. This exponentially increases the conditional probability table of each node and the joint probabilities to be calculated, thus, the system might become slow and unable to give answers within an acceptable time. Bayesian networks support continuous nodes with CPDs (Conditional Probability Densities) instead of CPTs and hybrid environments (mixed discrete and continuous nodes), which it would be nice to test and see how it performs in server utilisation predictions. However, this research field looks like a rather new one with limitations to specific distributions (mostly Gaussian), and not so much information, examples and implementations of software found in comparison to the discrete networks.

6.3.2 Speed Improvements

The current implementation is slow. The predictions for one node takes about 4.05 seconds every time the R script is called from Bayllocator. This can be improved significantly if one considers that the R library loading takes 0.65 seconds each time a prediction is called, the loading of the training dataset from file (which contains 130000 lines of data, and increasing for each guest at the time of writing) takes 0.6 seconds, 1.7 seconds takes the extraction of the CPTs for each node using the training dataset and 0.2 seconds the compilation of the CPTs for all of the nodes. From a quick calculation, only 4.05 – (0.65 + 0.6 + 1.7 + 0.2) = 0.9 seconds which is 22.2% of the total execution time is needed for the actual prediction. One way to improve the prediction speed with the current solution (by using R) is to use Rserve[2] [77] which is a resident R instance and it will only have to load the training data set and compile the CPTs once per day. Of course, trying other R packages or programming languages (Octave, Matlab) might be helpful, as it was proved for example that bnlearn would need much more time (more than 10 seconds) to execute the same query where gRain needs 0.9 seconds.

6.3.3 Prediction Improvements

The Bayesian networks created work individually for each virtual machine. One Bayesian network could be made for all of them, that it could even correlate servers affecting other servers. For example, a web server which is highly active and performing queries in a different database server, could be a causal node for the predictions of the database server. Also the date related variables used, proved not to be the best choice (for instance when it came to predict unexpected memory load) because the date is not the true reason behind high memory activity. The true reasons are probably the number of logged in users and the number of running processes or network connections. The date might be an indirect causal node since it affects the number of connected users in a

[2] http://stackoverflow.com/questions/9936116/r-script-and-library-preloading

server. Not well directed nodes and excessive number of states, decrease the learning efficiency of the Bayesian networks as demonstrated in section 5.1.7.

The current implementation does not deal with missing data imputation and gRain or bnlearn do not support it either. Email contact with the authors of both of these software products confirmed this. Since there was not much missed data, this did not affect the experiments. Some tests with similar training datasets containing a few missing data per column were made with gRain (without missing data imputation) and SamIam (which support EM learning) but the results were the same. At least a simple EM learning algorithm should be implemented.

6.3.4 Other Uses

Target group applications could be much wider than the memory allocation presented in this thesis. Any kind of workload could be predicted with Bayesian networks, and with more sophisticated design they could even be used for business related awareness like future server capacity planning if a system like this was trained to predict workload for a large data center. The modular nature of the system gives it much freedom and it can be build up and improved slowly as long as new correlations are observed from the trainer. This is something not easy to be achieved with traditional prediction methods.

6.4 Dynamic Memory Allocation Using Bayllocator

The dynamic memory allocation (resource in general) shows that it is very important to offer high quality of service in consolidated environments. A simple benchmark performed in section 5.2, shows the performance of a system can be devastated if its hardware resources are too small. Of course, when working in a single server as in this thesis, there might be cases when memory is not sufficient to serve all of the virtual machines, and swapping cannot be avoided. A workaround for this case, as it cannot be called a solution, could be an importance ranking of the guests running in the same hypervisor, so the lower ranked guests will be the first victims to be taken memory from.

The focus of this thesis was in the predictions and proactivity which are followed blindly by Bayllocator. Proactivity solve some problems of reactivity as it might be slow to sudden large memory changes, but reactivity is mandatory in a dynamic resource allocation system because if the prediction is wrong, then the system will perform very inadequately. To summarize, proactivity is needed for large memory changes (so it has to be good but not exact), while reactivity is needed for fine tuning and small changes. Using the Bayesian networks some typical reactivity was provided as described in section 5.1.5, proving also the wide purpose use of this different approach to the problem. The network in figure 5.22, could be a good candidate to provide both reactive and

proactive results by querying the VarFMU and PerFMU nodes respectively. Then Bayllocator could be programmed to act according to these numbers.

6.5 Future Work

Many more things could have been implemented and tested, but due to lack of time they had to be put in a future work list in this section.

- Collect different kind of data from servers with larger memory variation and activity, running different kind of real world applications like email server, web server, SIP server, ticketing system server from a large organisation where the people might be working during the day, SAN server etc. Run simulations and predictions on this data which will be more diverse than the generated or the collected data so far from the two real servers.
- Try the Bayesian structure learning of bnlearn package (or any other) on different data where correlations are not obvious (or even if they are), to see what kind of results it gives and how well are the predictions based on such a kind of network.
- Study and test hybrid Bayesian networks with discrete and continuous nodes, to achieve possibly higher prediction resolution.
- Create Bayesian networks to predict other kind of system utilisation (Disk I/O, CPU, Network I/O etc) and try to dynamically reallocate more than just memory.
- Create a Bayesian network to include more than one guests that they might affect each other (for example a web server making many queries in a database server)
- Polishing, and prediction speed improvement of the current implementation.
- Implementation of an algorithm to deal with data imputation in the training procedure.
- Run Bayllocator in a non-critical production environment and see how it behaves.
- Introduce reactivity, using a policy driven by the predictions.
- The only publicly available solution for KVM dynamic memory allocation found by the time of writing is MOM (Memory Overcommitment Manager) which was mentioned in the motivation section 1.1. Run MOM which only supports reactive ballooning (at the moment of writing), and see how it behaves with sudden increase of memory load. Compare with the proactivity of Bayllocator.

Chapter 7

Conclusion

The problem statement for this thesis was to create a proactive probabilistic self learning dynamic resource scaling system, focused in memory overcommitment using ballooning, and control the balloons with input from a Bayesian network. A prototype system, Bayllocator, was built to accomplish the task and the tests made against virtual machines running on a KVM hypervisor gave satisfactory results. Bayllocator, as any prototype needs further testing and improvements, but a good overview of how to use Bayesian networks to predict memory utilisation and the positive impacts of dynamic memory allocation in a highly consolidated environment presented. All of the initial questions have been answered, but through the journey more have been created which have been put to the future work section. In the following enumerated list, an answer to the corresponding questions made in section 1.2 is supplied.

1. A good Bayesian network, needs plenty of knowledge by observations. Significant data collection from different variables affecting the system might be needed, and good insight of how these variables are affecting each other directly or indirectly so that their causal relationships can be created. For memory predictions, good candidate variables can be considered to be the variables having direct impact to memory utilisation. These can be the number of processes running, users logged in, network utilisation and memory in use when the mentioned conditions are observed.

2. The collected data and variables can be anything with a potential to be encoded into useful information for the system. This can be raw data collected, or manipulated data (for example the rate of total memory as used in this thesis) before they will pipelined to the Bayesian network. The data will only have to be quantized properly in discrete states before it will be fed to the system.

3. As long as the nodes are well targeted and there is enough data for the training procedure, the system can give answers. Even for the randomly generated data in this thesis, the system was able to make accurate pre-

dictions.

4. The learning procedure was much simpler than the initial thoughts. When the Bayesian network structure exists, data which is collected has to be sent to the system, and as long as it receives new data-observations the system will improve its knowledge and answers. Once every day the training dataset is updated in this thesis.

5. The answer of the Bayesian network is always given in probabilities for specific states of an event represented by a node to happen, provided some optional evidence which will improve the answer. If there are 5 possible states in a query node, then 5 probabilities (1 for each state) will be provided. The user then is responsible to interpret these probabilities and make decisions. For the specific case of Bayllocator, all of them are used to form the expected memory as explained in section 4.3.3.2.

Bibliography

[1] S. Yates, E. Daley, B. Gray, J. Gownder, and R. Batiancila. Worldwide pc adoption forecast, 2007 to 2015. *Forrester Research Report*, 2007.

[2] Inc. Gartner. Press release: Gartner says more than 1 billion pcs in use worldwide and headed to 2 billion units by 2014. http://www.gartner.com/it/page.jsp?id=703807, Jun 2008. [Online; accessed 17-February-2012].

[3] L. Abramovsky and R. Griffith. Ict, corporate restructuring and productivity. 2009.

[4] D. Meisner, B.T. Gold, and T.F. Wenisch. Powernap: eliminating server idle power. *ACM SIGPLAN Notices*, 44(3):205–216, 2009.

[5] W. Vogels. Beyond server consolidation. *Queue*, 6(1):20–26, 2008.

[6] S.M. Brasol. *Analysis Of Advantages And Disadvantages To Server Virtualization*. PhD thesis, Bowie State University, 2009.

[7] Inc. Gartner. Forecast: Hosted virtual desktops, worldwide, 2010-2014 (2010 update), G00206728, Dec 2010.

[8] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on*, pages 119–128. IEEE, 2007.

[9] M. Schmid, D. Marinescu, and R. Kroeger. A framework for autonomic performance management of virtual machine-based services. In *Proceedings of the 15th Annual Workshop of HP Software University Assocaiton. Hosted by AI Akhawayn University in Ifran, June*, pages 22–25, 2008.

[10] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

[11] A. Arcangeli, I. Eidus, and C. Wright. Increasing memory density by using ksm. In *Proceedings of the Linux Symposium*, pages 19–28, 2009.

[12] W. Zhou, S. Yang, J. Fang, X. Niu, and H. Song. Vmctune: A load balancing scheme for virtual machine cluster using dynamic resource allocation. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 81–86. Ieee, 2010.

[13] G. Kukreja and S. Singh. Virtio based transcendent memory. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 1, pages 723–727. IEEE, 2010.

[14] C.A. Waldspurger. Memory resource management in vmware esx server. *ACM SIGOPS Operating Systems Review*, 36(SI):181–194, 2002.

[15] J.H. Schopp, K. Fraser, and M.J. Silbermann. Resizing memory with balloons and hotplug. In *Proceedings of the Linux Symposium*, volume 2, pages 313–319, 2006.

[16] Xen. Xen celebrates full dom0 and domu support in linux 3.0. http://blog.xen.org/index.php/2011/06/02/xen-celebrates-full-dom0-and-domu-support-in-linux-3-0/, June 2011. [Online; accessed 19-February-2012].

[17] Canonical. Ubuntu picks KVM over XEN for virtualization. http://www.linux-kvm.com/content/ubuntu-picks-kvm-over-xen-virtualization, Feb 2008. [Online; accessed 19-February-2012].

[18] Red Hat. Red hat advances virtualization leadership with qumranet, inc. acquisition. http://www.redhat.com/about/news/press-archive/2008/9/qumranet, Sep 2008. [Online; accessed 19-February-2012].

[19] Taneja Group. Hypervisor shootout: Maximizing workload density in the virtualization platform. http://tanejagroup.com/files/Hypervisor-Shootout-Maximizing-Workload1.pdf, Aug 2010. [Online; accessed 17-February-2012].

[20] Lamont Granquist. Kvm vs. xenserver vs. vmware memory overcommitment. http://www.scriptkiddie.org/blog/2010/06/27/kvm-vs-xenserver-vs-vmware-memory-overcommitment/, Jun 2010. [Online; accessed 01-March-2012].

[21] Nick Anderson. Memory overcommitment in xenserver and esx. http://speakvirtual.com/2011/02/28/memory-overcommitment-in-xenserver-and-esx/, Feb 2011. [Online; accessed 01-March-2012].

[22] vDoppler.net. Dynamic memory and xenserver. http://vdoppler.net/?p=145, Dec 2010. [Online; accessed 01-March-2012].

[23] Adam Litke. Managing resources on over-committed virtualization hosts. In *KVM Forum*, 2010.

[24] W. Feng, F. Chang, W. Feng, and J. Walpole. Provisioning on-line games: a traffic analysis of a busy counter-strike server. In *Proceedings of the 2nd*

ACM SIGCOMM Workshop on Internet measurment, pages 151–156. ACM, 2002.

[25] Y. Baryshnikov, E. Coffman, G. Pierre, D. Rubenstein, M. Squillante, and T. Yimwadsana. Predictability of web-server traffic congestion. In *Web Content Caching and Distribution, 2005. WCW 2005. 10th International Workshop on*, pages 97–103. IEEE, 2005.

[26] R.K. Sahoo, A.J. Oliner, I. Rish, M. Gupta, J.E. Moreira, S. Ma, R. Vilalta, and A. Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–435. ACM, 2003.

[27] E. Charniak. Bayesian networks without tears. *AI magazine*, 12(4):50, 1991.

[28] Wikipedia. Bayesian network — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Bayesian_network&oldid=479309760, 2012. [Online; accessed 1-March-2012].

[29] Wikipedia. Artificial intelligence — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Artificial_intelligence&oldid=479075069, 2012. [Online; accessed 1-March-2012].

[30] D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. *Games and Economic Behavior*, 45(1):181–221, 2003.

[31] R.M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Verlag, 1996.

[32] D. Kersten, P. Mamassian, and A. Yuille. Object perception as bayesian inference. *Annu. Rev. Psychol.*, 55:271–304, 2004.

[33] J.M.L. Sloughter, A.E. Raftery, T. Gneiting, and C. Fraley. Probabilistic quantitative precipitation forecasting using bayesian model averaging. 2010.

[34] A.S. Cofıño, R. Cano, C. Sordo, and J.M. Gutiérrez. Bayesian networks for probabilistic weather prediction. In *15th Eureopean Conference on Artificial Intelligence, ECAI*. Citeseer, 2002.

[35] PJ Lucas, L.C. van der Gaag, A. Abu-Hanna, et al. Bayesian networks in biomedicine and health-care. *Artificial Intelligence in medicine*, 30(3):201, 2004.

[36] G. Zweig and S. Russell. Speech recognition with dynamic bayesian networks. 1998.

[37] J.A. Zdziarski. *Ending spam: Bayesian content filtering and the art of statistical language classification*. No Starch Press, 2005.

[38] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers*

from the 1998 workshop, volume 62, pages 98–105. Madison, Wisconsin: AAAI Technical Report WS-98-05, 1998.

[39] Spam Assassin. Bayes in spam assassin. http://wiki.apache.org/spamassassin/BayesInSpamAssassin, Mar 2012. [Online; accessed 01-March-2012].

[40] A.W. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 50–60. ACM, 2005.

[41] S. Khan, J. Loo, J.L. Mauri, and J.H. Ortiz. *Mobile Ad Hoc Networks: Current Status and Future Trends*. Taylor & Francis, 2011.

[42] S.R. Ellis. Nature and origins of virtual environments: a bibliographical essay. *Computing Systems in Engineering*, 2(4):321 – 347, 1991.

[43] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39:50–55, December 2008.

[44] Hanfei Dong, Qinfen Hao, Tiegang Zhang, and Bing Zhang. Formal discussion on relationship between virtualization and cloud computing. *Parallel and Distributed Computing Applications and Technologies, International Conference on*, 0:448–453, 2010.

[45] J.E. Smith and Ravi Nair. The architecture of virtual machines. *Computer*, 38(5):32 – 38, may 2005.

[46] Mladen A. Vouk. Cloud Computing - Issues, Research and Implementations. *Journal of Computing and Information Technology*, 16(4):235–246, 2008.

[47] Edward Ray and Eugene Schultz. Virtualization security. In *Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research: Cyber Security and Information Intelligence Challenges and Strategies*, CSIIRW '09, pages 42:1–42:5, New York, NY, USA, 2009. ACM.

[48] Kirk L. Kroeker. The evolution of virtualization. *Commun. ACM*, 52:18–20, March 2009.

[49] B. Yamini and D.V. Selvi. Cloud virtualization: A potential way to reduce global warming. In *Recent Advances in Space Technology Services and Climate Change (RSTSCC), 2010*, pages 55 –57, nov. 2010.

[50] Liang Liu, Hao Wang, Xue Liu, Xing Jin, Wen Bo He, Qing Bo Wang, and Ying Chen. Greencloud: a new architecture for green data center. In *Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*, ICAC-INDST '09, pages 29–38, New York, NY, USA, 2009. ACM.

[51] Wikipedia. Snapshot (computer storage) — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Snapshot

_(computer_storage)&oldid=478410432, 2012. [Online; accessed 2-March-2012].

[52] K. Miller and M. Pegah. Virtualization: virtually at the desktop. In *Proceedings of the 35th annual ACM SIGUCCS fall conference*, pages 255–260. ACM, 2007.

[53] J. Rutkowska and R. Wojtczuk. Qubes os architecture. *Invisible Things Lab, Tech. Rep*, 2010.

[54] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37:164–177, October 2003.

[55] KVM. Kvm-75 gets ballooning. http://www.linux-kvm.org/page/ChangeLog, Sep 2008. [Online; accessed 2-March-2012].

[56] W. Hwang, Y. Roh, Y. Park, K.W. Park, and K.H. Park. Hyperdealer: Reference-pattern-aware instant memory balancing for consolidated virtual machines. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 426–434. IEEE, 2010.

[57] T. Wood, G. Tarasuk-Levin, P. Shenoy, P. Desnoyers, E. Cecchet, and M.D. Corner. Memory buddies: exploiting page sharing for smart colocation in virtualized data centers. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 31–40. ACM, 2009.

[58] W. Zhao, Z. Wang, and Y. Luo. Dynamic memory balancing for virtual machines. *ACM SIGOPS Operating Systems Review*, 43(3):37–47, 2009.

[59] D. Williams, H. Jamjoom, Y.H. Liu, and H. Weatherspoon. Overdriver: Handling memory overload in an oversubscribed cloud. In *Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 205–216. ACM, 2011.

[60] D. Heckerman. A tutorial on learning with bayesian networks. *Innovations in Bayesian Networks*, pages 33–82, 2008.

[61] E. Lindfors. Bayesian inference–a way to combine statistical data and semantic analysis meaningfully. *Techne Series: Research in Sloyd Education and Craft Science A*, 18(1), 2011.

[62] Wikipedia. Bayes' theorem — wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Bayes4665, 2012. [Online; accessed 12-March-2012].

[63] T.A. Stephenson. An introduction to bayesian network theory and usage. *IDIAP Research Report 00-03*, 2000.

[64] Adnan Darwiche. Samiam. http://reasoning.cs.ucla.edu/samiam/, 2012. [Online; accessed 12-March-2012].

[65] Y. Wu, J. McCall, and D. Corne. Two novel ant colony optimization approaches for bayesian network structure learning. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–7. IEEE, 2010.

[66] R. Daly. Using ant colony optimization in learning bayesian network equivalence classes. 2006.

[67] X. Dong, H.H. Yu, D. Ouyang, D. Cai, Y. Ye, and Y. Zhang. Cooperative coevolutionary genetic algorithms to find optimal elimination orderings for bayesian networks. In *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on*, pages 1388–1394. IEEE, 2010.

[68] K.J. Kim and S.B. Cho. Evolutionary aggregation and refinement of bayesian networks. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1513–1520. IEEE, 2006.

[69] Y. Wu, J. McCall, and D. Corne. Comparative analysis of search and score metaheuristics for bayesian network structure learning using node juxtaposition distributions. *Parallel Problem Solving from Nature–PPSN XI*, pages 424–433, 2011.

[70] P. Padala, K.G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. *ACM SIGOPS Operating Systems Review*, 41(3):289–302, 2007.

[71] Libvirt. http://libvirt.org/, 2012. [Online; accessed 02-May-2012].

[72] P.G. Bringas. *Database and Expert Systems Applications: 21st International Conference, DEXA 2010, Bilbao, Spain, August 30-September 3, 2010, Proceedings*, volume 2. Springer-Verlag New York Inc, 2010.

[73] Marco Scutari. Learning bayesian networks with the bnlearn R package. *Journal of Statistical Software*, 35(3):1–22, 2010.

[74] Søren Højsgaard. Graphical independence networks with the gRain package for R. *Journal of Statistical Software*, 46(10):1–26, 2012.

[75] J.R. Hauser. *Numerical methods for nonlinear engineering models*. Springer Verlag, 2009.

[76] A. DasGupta. *Probability for Statistics and Machine Learning: Fundamentals and Advanced Topics*. Springer Verlag, 2011.

[77] Rserve. http://rforge.net/Rserve/, 2012. [Online; accessed 15-May-2012].

Appendix A

Scripts

Listing A.1: bayllocator.pl

```
#!/usr/bin/perl

# Vangelis Tasoulas <evanget@ifi.uio.no>

# Bayllocator -- Dynamically change memory on KVM Virtual Machines

# Needed packages
use lib '/usr/local/lib/bayllocator';
use lib 'lib';
use lib '../lib';
use strict 'vars';
use Getopt::Std;
use feature 'say';
use Time::Local;
use Time::HiRes qw(sleep);
use POSIX qw(strftime floor ceil);
use IO::Socket::UNIX qw( SOCK_STREAM );
use JSON;
use Clone qw( clone );
use Sys::Virt;
use MyLibVirtSubs;
use MyTimeSubs;
use List::Util qw(sum);
use Data::Dumper;
use MyConfig;
use MyEMailNotifier;
use MyBayesianPredictor;

# Global variables
my $VERBOSE = 0;
my $DEBUG = 0;
my $IGNORE = 0;

my $PREDICTOR_BIN = "predictor.pl";

#####################
# handle flags and arguments
# Example: c == "-c", c: == "-c argument"
my $opt_string = 'hvdi';
getopts( "$opt_string", \my %opt ) or usage() and exit 1;

# print help message if -h is invoked
if ( $opt{'h'} ) {
```

```
44    usage();
45    exit 0;
46  }
47
48  $VERBOSE = 1 if $opt{'v'};
49  $DEBUG = 1 if $opt{'d'};
50  $IGNORE = 1 if $opt{'i'};
51
52  my $virtualMachineManager = Sys::Virt->new( address => $config{virsh_host} );
53
54  ################################
55  ################################
56  ################################
57  # Main program content #
58  ################################
59  ################################
60  ################################
61
62  my %GUEST;
63
64  my $RESULT = delay_and_get_current_mem_5_samples_avg(0);
65  foreach ( keys %{$RESULT} ) {
66    $GUEST{$_}{CurrentMem5samplesAvg} = $$RESULT{$_}{CurrentMem5samplesAvg};
67    $GUEST{$_}{PreviousMem5samplesAvg} = $GUEST{$_}{CurrentMem5samplesAvg};
68  }
69
70  while (1) {
71
72    my $loopStart = [ Time::HiRes::gettimeofday() ];
73
74    my $total_mem_predicted = 0;
75    my $total_guests_running = 0;
76    my $minimum_Hypervisor_memory = 0;
77
78    # Time to ask for predictions (Somewhere in the future)
79    my $timestamp_to_get_answer_for = time + $config{predict_future};
80
81    # This foreach loop will read the current actual memory from each guest
82    # and make the predictions for the future.
83    foreach my $dom ( get_active_domains($virtualMachineManager) ) {
84      $total_guests_running++;
85      my $domUuid = $dom->get_uuid_string;
86      my $domName = $dom->get_name;
87      my $domInfo = $dom->get_info;
88
89      debug("---- $domName ----");
90
91      $GUEST{$domUuid}{name} = $domName;
92      $GUEST{$domUuid}{Guest_MaxMemory} = $$domInfo{maxMem};
93      $GUEST{$domUuid}{Guest_Memory} = $$domInfo{memory};
94      $GUEST{$domUuid}{memChange_due_to_limitations} = 0;
95
96      debug( "Guest_Memory: " . $GUEST{$domUuid}{Guest_Memory} );
97
98      debug( "Time now: " . timestamp_to_datetime(time) );
99      debug( "Prediction made for: " . timestamp_to_datetime($timestamp_to_get_answer_for) );
100
101     my $RESULT = get_guest_memory( $domUuid, $domName );
102     foreach my $key ( keys %{ $$RESULT{$domUuid} } ) {
103       $GUEST{$domUuid}{$key} = $$RESULT{$domUuid}{$key};
104     }
105
106     # In memory overcommitment, the hypervisor will introduce some overhead which is a fixed ↙
            ↳ number related
107     # to the max given memory. We can get the overhead by subtracting the virtual machine actual ↙
            ↳ memory from
```

```
108     # the memory given by the hypervisor to the virtual machine.
109     # Example: If maxMem is set to 4194304KB and the guest memory is set to 1048576KB, the actual ↙
            ↳ memory
110     # the virtual machine sees is 910712KB. The overhead then is 1048576KB – 910712KB = 137864KB
111     $GUEST{$domUuid}{Hypervisor_Memory_Overhead} = $GUEST{$domUuid}{Guest_Memory} ↙
            ↳ – $GUEST{$domUuid}{memTotal};
112     $GUEST{$domUuid}{Guest_MinMemory} = $config{minimum_guest_memory} + ↙
            ↳ $GUEST{$domUuid}{Hypervisor_Memory_Overhead};
113     debug( "Hypervisor_Memory_Overhead: " . ↙
            ↳ $GUEST{$domUuid}{Hypervisor_Memory_Overhead} );
114     debug( "Guest_MinMemory: " . $GUEST{$domUuid}{Guest_MinMemory} );
115
116     $minimum_Hypervisor_memory += $GUEST{$domUuid}{Guest_MinMemory};
117
118     my $result = bayesian_predictor( $timestamp_to_get_answer_for, ↙
            ↳ $GUEST{$domUuid}{CurrentMem5samplesAvg}, $domName );
119     $GUEST{$domUuid}{PreviousMem5samplesAvg} = ↙
            ↳ $GUEST{$domUuid}{CurrentMem5samplesAvg};
120     if ( $result == -1 ) {
121       warn "Unexpected error in prediction.";
122     } else {
123       $GUEST{$domUuid}{mem_predicted} = $result;
124     }
125
126     debug( "Predicted: " . $GUEST{$domUuid}{mem_predicted} );
127
128     open( PREDFILE, ">>/home/evanget/BayllocatorPredictions.log" ) or warn "$!";
129     say PREDFILE $GUEST{$domUuid}{name} . "," . time . ",$timestamp_to_get_answer_for," . ↙
            ↳ $GUEST{$domUuid}{mem_predicted};
130     close(PREDFILE);
131
132     # Set $GUEST{$domUuid}{mem_to_set} to the predicted value plus memory overhead plus the ↙
            ↳ $config{add_to_prediction}
133     $GUEST{$domUuid}{mem_to_set} = sprintf( "%0.f", ( $GUEST{$domUuid}{mem_predicted} + ( ↙
            ↳ $GUEST{$domUuid}{mem_predicted} * $config{add_to_prediction} ) ) + ↙
            ↳ $GUEST{$domUuid}{Hypervisor_Memory_Overhead} );
134     if ( $GUEST{$domUuid}{mem_to_set} > $GUEST{$domUuid}{Guest_MaxMemory} ) {
135       my $message = "Guest $GUEST{$domUuid}{name} needs more memory than the Maximum ↙
              ↳ Allowed\n";
136       $message .= "Needs: $GUEST{$domUuid}{mem_to_set}\n";
137       $message .= "Max: $GUEST{$domUuid}{Guest_MaxMemory}";
138       warn $message;
139       send_email_to_admins( $message, 'warn' ) if ( !$IGNORE );
140     }
141     $total_mem_predicted += $GUEST{$domUuid}{mem_to_set};
142
143     debug("-----------------------------------------------\n");
144
145   }
146
147   # Get a percentage of how much memory each guest will be using out of the predicted amount.
148   foreach my $guest ( keys %GUEST ) {
149     $GUEST{$guest}{percent_of_total_predicted} = $GUEST{$guest}{mem_to_set} / ↙
            ↳ $total_mem_predicted;
150   }
151
152   # Get memory information for the Hypervisor
153   my $hypervisor_mem = get_hypervisor_memory();
154
155   # $memory_available_for_guests is the variable which will make sure that the
156   # Hypervisor will use a fixed amount of memory for the virtual machines.
157   # This will not let the system swap in excess demand and it will give memory
158   # space for more applications the hypervisor might be running.
159   my $memory_available_for_guests = $$hypervisor_mem{memTotal} – $config{hypervisor_memory};
160
161   my $message = "You are using less than 50% of the hypervisor's total memory for your guests.\n";
```

```
162   $message .= "You might want to decrease the value of \'\$config{hypervisor_memory}\' in the ↙
          ↳ configuration file.";
163   warn($message) if ( ( $memory_available_for_guests / $$hypervisor_mem{memTotal} ) < 0.50 and ↙
          ↳ !$IGNORE );
164
165   my $message = "Hypervisor does not have enough memory to serve your minimum requirements.\n";
166   $message .= "Available for virtual machines: $memory_available_for_guests\n";
167   $message .= "Minimum needs of your virtual machines: $minimum_Hypervisor_memory\n";
168   $message .= "You might want to decrease the value of \'\$config{hypervisor_memory}\' in the ↙
          ↳ configuration file.";
169   if ( ( $memory_available_for_guests < $minimum_Hypervisor_memory ) and !$IGNORE ) {
170     send_email_to_admins( $message, 'fatal' );
171     die($message);
172   }
173
174   verbose( "Hypervisor total memory: " . sprintf( "%0.f", $$hypervisor_mem{memTotal} / 1024 ) . " ↙
          ↳ MB, " . $$hypervisor_mem{memTotal} . " KB" );
175   verbose( "Memory available for guests: " . sprintf( "%0.f", $memory_available_for_guests / 1024 ) . " ↙
          ↳ MB, " . $memory_available_for_guests . " KB" );
176   verbose("-----------------------------------------------\n");
177
178   verbose("Total Memory Predicted to use: $total_mem_predicted KB");
179
180   my $extra_mem_available = 0;
181   if ( $total_mem_predicted < $memory_available_for_guests ) {
182
183     $extra_mem_available = floor( $memory_available_for_guests - $total_mem_predicted );
184     verbose("Extra Memory Available in addition to the predicted for the guests: $extra_mem_available ↙
            ↳ KB");
185     verbose("This will be distributed to the virtual machines according to their demand.");
186
187   } elsif ( $total_mem_predicted > $memory_available_for_guests ) {
188
189     my $message = "The total amount of memory needed for the guests exceeds the Maximum ↙
            ↳ Allowed to be used.\n";
190     $message .= "Needs: $total_mem_predicted\n";
191     $message .= "Max: $memory_available_for_guests";
192     warn $message;
193     send_email_to_admins( $message, 'warn' ) if ( !$IGNORE );
194
195     $extra_mem_available = ceil( $total_mem_predicted - $memory_available_for_guests );
196     verbose("Extra Memory Needed for the guests: $extra_mem_available KB\n");
197
198   }
199
200   verbose(" ");
201
202   my $policy_violation = 0;
203
204   verbose("---- Calculate Memory to set ----");
205
206   # Add or remove memory to the guests if hypervisor's memory is
207   # underutilised or overutilised.
208   foreach my $guest ( keys %GUEST ) {
209
210     verbose("---- $GUEST{$guest}{name} ----");
211
212     # If the total predicted memory to use is less than the
213     # memory available from the hypervisor for the guests, then
214     # instead of wasting this memory, distribute it to the virtual
215     # machines given the demand for each of them (percent_of_total_predicted).
216     # Extra memory (if available) always boosts performance due to
217     # disk caching reasons.
218     # If the available is less than the needed, remove memory from the
219     # virtual machines
220     verbose("Calculated: $GUEST{$guest}{mem_to_set} KB");
```

```
221     if ( $total_mem_predicted < $memory_available_for_guests ) {
222
223       my $additional_mem = floor( $GUEST{$guest}{percent_of_total_predicted} * ↙
              ↳ $extra_mem_available );
224       verbose( "Additional Memory to set due to excess: " . $additional_mem . " KB" );
225       $GUEST{$guest}{mem_to_set} = $GUEST{$guest}{mem_to_set} + $additional_mem;
226       verbose("Calculated now: $GUEST{$guest}{mem_to_set} KB");
227
228     } elsif ( $total_mem_predicted > $memory_available_for_guests ) {
229
230       my $remove_mem = ceil( $GUEST{$guest}{percent_of_total_predicted} * $extra_mem_available );
231       verbose( "Memory to remove due to limitations: " . $remove_mem . " KB" );
232       $GUEST{$guest}{mem_to_set} = $GUEST{$guest}{mem_to_set} - $remove_mem;
233       verbose("Calculated now: $GUEST{$guest}{mem_to_set} KB");
234
235     }
236
237     $policy_violation = 1 if ( memory_to_set_meets_policy_requirements($guest) == 1 );
238
239     verbose("-------------------------------------------------\n");
240   }
241
242   while ($policy_violation) {
243     fix_policy_violations();
244     $policy_violation = 0;
245     foreach my $guest ( keys %GUEST ) {
246       $policy_violation = 1 if ( memory_to_set_meets_policy_requirements($guest) == 1 );
247     }
248   }
249
250   # Eventually, set the memory to the virtual machines.
251   foreach my $dom ( get_active_domains($virtualMachineManager) ) {
252
253     my $domUuid = $dom->get_uuid_string;
254
255     verbose("---- $GUEST{$domUuid}{name} ----");
256     verbose( "Memory set: " . $GUEST{$domUuid}{mem_to_set} . " KB" );
257     verbose( "Useful for the guest (due to Hypervisor memory overhead): " . ( ↙
            ↳ $GUEST{$domUuid}{mem_to_set} - ↙
            ↳ $GUEST{$domUuid}{Hypervisor_Memory_Overhead} ) . " KB" );
258     verbose("-------------------------------------------------\n");
259
260     $dom->set_memory( $GUEST{$domUuid}{mem_to_set}, Sys::Virt::Domain::MEM_LIVE );
261
262   }
263
264   my $timeNeededToRun = Time::HiRes::tv_interval($loopStart);
265
266   verbose( "Prediction made for: " . timestamp_to_datetime($timestamp_to_get_answer_for) );
267   verbose("\nElapsed time: $timeNeededToRun seconds!\n\n");
268
269   # Sleep until the next $config{action_interval} comes.
270   my $RESULT = delay_and_get_current_mem_5_samples_avg( $config{action_interval} - ↙
          ↳ $timeNeededToRun );
271   $GUEST{$_}{CurrentMem5samplesAvg} = $$RESULT{$_}{CurrentMem5samplesAvg} foreach ( ↙
          ↳ keys %{$RESULT} );
272
273 }
274
275 # End opf Main program
276
277 ################################
278 ################################
279 ################################
280 # Helper routines #
281 ################################
```

```
282 ##################################
283 ##################################
284
285 # memory_to_set_meets_policy_requirements will check if
286 # the memory to be set violates the policy requirements.
287 sub memory_to_set_meets_policy_requirements {
288
289     my $guest = $_[0];
290
291     my $violated_policy = 0;
292
293     # $GUEST{$domUuid}{memChange_due_to_limitations} variable keeps information for
294     # the extra (or less) memory needed by a virtual machine after the adjustments.
295     # For example if a virtual machine needs 100MB of memory and after the
296     # added $config{add_to_prediction} and $GUEST{$domUuid}{Hypervisor_Memory_Overhead}
297     # and any added or removed memory due to the extra $memory_available_for_guests
298     # still need 250MB, this violates the $GUEST{$domUuid}{Guest_MinMemory} that by default
299     # it is set at 300MB. This virtual machine then will take +50MB of the total calculated
300     # which we will have to take from another virtual machine to avoid host swapping.
301     # In a similar manner, if the calculated memory exceeds the $GUEST{$domUuid}{Guest_MaxMemory}
302     # This additional memory can be distributed to other virtual machines to get
303     # some performance benefit.
304     $GUEST{$guest}{memChange_due_to_limitations} = 0;
305
306     # If the memory to set is more than the maximum allowed for
307     # the guest, then set the Max memory to it.
308     if ( $GUEST{$guest}{mem_to_set} > $GUEST{$guest}{Guest_MaxMemory} ) {
309
310         # $GUEST{$domUuid}{memChange_due_to_limitations} will get a negative value in this case
311         $GUEST{$guest}{memChange_due_to_limitations} = $GUEST{$guest}{Guest_MaxMemory} - ↙
                ↪ $GUEST{$guest}{mem_to_set};
312         verbose( "Memory to set exceeds the maximum limit for the guest which is " . ↙
                ↪ $GUEST{$guest}{Guest_MaxMemory} . "." );
313         verbose( abs( $GUEST{$guest}{memChange_due_to_limitations} ) . " KB less memory will be set ↙
                ↪ to meet the Max Memory policy." );
314         $GUEST{$guest}{mem_to_set} = $GUEST{$guest}{Guest_MaxMemory};
315     }
316
317     # If the memory to set is less than the minimum allowed for
318     # the guest, then set the Min memory to it.
319     if ( $GUEST{$guest}{mem_to_set} < $GUEST{$guest}{Guest_MinMemory} ) {
320
321         # $GUEST{$domUuid}{memChange_due_to_limitations} will get a positive value in this case
322         $GUEST{$guest}{memChange_due_to_limitations} = $GUEST{$guest}{Guest_MinMemory} - ↙
                ↪ $GUEST{$guest}{mem_to_set};
323         verbose( "Memory to set is less than the minimum limit for the guest which is " . ↙
                ↪ $GUEST{$guest}{Guest_MinMemory} . "." );
324         verbose( $GUEST{$guest}{memChange_due_to_limitations} . "KB more memory will be set to ↙
                ↪ meet the Min Memory policy." );
325         $GUEST{$guest}{mem_to_set} = $GUEST{$guest}{Guest_MinMemory};
326     }
327
328     return 1 if $GUEST{$guest}{memChange_due_to_limitations} != 0;
329     return 0 if $GUEST{$guest}{memChange_due_to_limitations} == 0;
330
331 }
332
333 sub fix_policy_violations {
334
335     # Since we added or removed memory, some guests might have violated the
336     # Min and Max memory policies.
337     # This leads to more (or less) free memory than the initially calculated
338     # for the virtual machines not violating any of the policies.
339     # The violating guests will have a non zero value for the
340     # $GUEST{$guest}{memChange_due_to_limitations} variable.
341     #
```

```
342     # Now if we had three hosts with a total demand of 1.500MB
343     # 1st host needs 550MB | 36.66% of 1500MB
344     # 2nd host needs 250MB | 16.66% of 1500MB <- This one violates the minimum memory
345     # 3rd host needs 700MB | 46.66% of 1500MB
346     #
347     # The second host have already taken 50MB extra.
348     # We have to reduce this 50MB from the non violating hosts.
349     # These are the the 1st and 3rd hosts in this case.
350     # To calculate how many MB we will take from each of the 1st and 3rd host
351     # we need to take into consideration their prior percentage of the total
352     # memory ($GUEST{$guest}{percent_of_total_predicted}) and fit it to
353     # a new one as follows:
354     #
355     # 36.66% + 46.66% = 83.32%
356     # 36.66% / 83.32% = 44%
357     # 46.66% / 83.32% = 56%
358     # 44% * 50MB = 22MB
359     # 56% * 50MB = 28MB
360     #
361     # So after the fitting, we will take 22MB from the 1st host and
362     # 28MB from the 3rd host to avoid any hypervisor swapping.
363     my $total_memory_to_fit_after_policy_violations_taken_into_consideration = 0;
364     my $total_percentage_left_to_fit_after_policy_violations_taken_into_consideration = 0;
365     foreach my $guest ( keys %GUEST ) {
366       if ( $GUEST{$guest}{memChange_due_to_limitations} != 0 ) {
367         $total_memory_to_fit_after_policy_violations_taken_into_consideration += ↙
                ↳ $GUEST{$guest}{memChange_due_to_limitations};
368       } else {
369         $total_percentage_left_to_fit_after_policy_violations_taken_into_consideration += ↙
                ↳ $GUEST{$guest}{percent_of_total_predicted};
370       }
371     }
372
373     verbose(" ");
374
375     if ( $total_memory_to_fit_after_policy_violations_taken_into_consideration > 0 ) {
376       verbose("More memory has been acquired by virtual machines violating the minimum policy ↙
              ↳ requirements.");
377       verbose("The total memory acquired to fix the violations is ↙
              ↳ $total_memory_to_fit_after_policy_violations_taken_into_consideration KB and it will be ↙
              ↳ taken proportionally by non violating hosts\n");
378     } elsif ( $total_memory_to_fit_after_policy_violations_taken_into_consideration < 0 ) {
379       verbose("Less memory has been acquired by virtual machines tried to acquire more than the ↙
              ↳ maximum allowed.");
380       verbose( "The extra memory available by fixing the violations is " . ↙
              ↳ abs($total_memory_to_fit_after_policy_violations_taken_into_consideration) . " KB and it ↙
              ↳ will be given proportionally to non violating hosts\n" );
381     }
382
383     foreach my $guest ( keys %GUEST ) {
384
385       # If this is > 0, then it means that we would have used more memory
386       # for virtual machines given less than the Minimum allowed memory.
387       # So this amount of memory has to be taken from virtual machines not
388       # violating any policies (min or max)
389       if ( $total_memory_to_fit_after_policy_violations_taken_into_consideration > 0 ) {
390
391         if ( $GUEST{$guest}{memChange_due_to_limitations} == 0 ) {
392           $GUEST{$guest}{percent_after_fitting} = $GUEST{$guest}{percent_of_total_predicted} / ↙
                  ↳ $total_percentage_left_to_fit_after_policy_violations_taken_into_consideration;
393           my $mem_remove = ceil( $GUEST{$guest}{percent_after_fitting} * ↙
                  ↳ $total_memory_to_fit_after_policy_violations_taken_into_consideration );
394           $GUEST{$guest}{mem_to_set} = $GUEST{$guest}{mem_to_set} - $mem_remove;
395           verbose("Removed $mem_remove KB of memory from guest $GUEST{$guest}{name}");
396         }
397
```

```
398     } elsif ( $total_memory_to_fit_after_policy_violations_taken_into_consideration < 0 ) {
399
400       if ( $GUEST{$guest}{memChange_due_to_limitations} == 0 ) {
401         $GUEST{$guest}{percent_after_fitting} = $GUEST{$guest}{percent_of_total_predicted} / ↙
                ↳ $total_percentage_left_to_fit_after_policy_violations_taken_into_consideration;
402         my $mem_add = floor( $GUEST{$guest}{percent_after_fitting} * ↙
                ↳ abs($total_memory_to_fit_after_policy_violations_taken_into_consideration) );
403         $GUEST{$guest}{mem_to_set} = $GUEST{$guest}{mem_to_set} + $mem_add;
404         verbose("Added $mem_add KB of memory to guest $GUEST{$guest}{name}");
405       }
406
407     }
408   }
409
410   verbose("\n");
411 }
412
413 sub get_hypervisor_memory {
414
415   my $file = "/proc/meminfo";
416
417   my %MEM;
418
419   open FILE, $file or die $!;
420   while ( my $line = <FILE> ) {
421     chomp($line);
422     $MEM{memTotal} = $1 if $line =~ /^MemTotal:\s+(\d+)/;
423     $MEM{memUnused} = $1 if $line =~ /^MemFree:\s+(\d+)/;
424     $MEM{memBuffers} = $1 if $line =~ /^Buffers:\s+(\d+)/;
425     $MEM{memCached} = $1 if $line =~ /^Cached:\s+(\d+)/;
426     $MEM{swapTotal} = $1 if $line =~ /^SwapTotal:\s+(\d+)/;
427     $MEM{swapFree} = $1 if $line =~ /^SwapFree:\s+(\d+)/;
428     $MEM{swapCached} = $1 if $line =~ /^SwapCached:\s+(\d+)/;
429   }
430   close FILE;
431   $MEM{memActuallyFree} = $MEM{memUnused} + $MEM{memBuffers} + $MEM{memCached};
432   $MEM{memActuallyUsed} = $MEM{memTotal} - $MEM{memActuallyFree};
433   $MEM{swapActuallyFree} = $MEM{swapFree} + $MEM{swapCached};
434   $MEM{swapActuallyUsed} = $MEM{swapTotal} - $MEM{swapActuallyFree};
435
436   return \%MEM;
437
438 }
439
440 sub get_guest_memory {
441
442   my $domUuid = $_[0];
443   my $domName = $_[1];
444   my %MEM;
445
446   my $socket_path = '/tmp/' . $domUuid . '-2.qemuga.sock';
447
448   my $socket = new IO::Socket::UNIX(
449     Type => SOCK_STREAM,
450     Peer => $socket_path
451   ) or warn "Can't connect to socket $socket_path: $!\n";
452
453   if ( defined $socket ) {
454
455     my $meminfo = -1;
456
457     $MEM{$domUuid}{memTotal} = 0;
458     $MEM{$domUuid}{memUnused} = 0;
459     $MEM{$domUuid}{memBuffers} = 0;
460     $MEM{$domUuid}{memCached} = 0;
461     $MEM{$domUuid}{swapTotal} = 0;
```

```
462     $MEM{$domUuid}{swapFree} = 0;
463     $MEM{$domUuid}{swapCached} = 0;
464
465     $MEM{$domUuid}{memActuallyFree} = 0;
466     $MEM{$domUuid}{memActuallyUsed} = 0;
467     $MEM{$domUuid}{swapActuallyFree} = 0;
468     $MEM{$domUuid}{swapActuallyUsed} = 0;
469
470     while ( $meminfo == -1 ) {
471
472       qemuga_syncguest($socket);
473
474       $meminfo = qemuga_readfile( $socket, "/proc/meminfo" );
475
476       my @lines = split /\n/, $meminfo;
477       foreach my $line (@lines) {
478         $MEM{$domUuid}{memTotal} = $1 if $line =~ /^MemTotal:\s+(\d+)/;
479         $MEM{$domUuid}{memUnused} = $1 if $line =~ /^MemFree:\s+(\d+)/;
480         $MEM{$domUuid}{memBuffers} = $1 if $line =~ /^Buffers:\s+(\d+)/;
481         $MEM{$domUuid}{memCached} = $1 if $line =~ /^Cached:\s+(\d+)/;
482         $MEM{$domUuid}{swapTotal} = $1 if $line =~ /^SwapTotal:\s+(\d+)/;
483         $MEM{$domUuid}{swapFree} = $1 if $line =~ /^SwapFree:\s+(\d+)/;
484         $MEM{$domUuid}{swapCached} = $1 if $line =~ /^SwapCached:\s+(\d+)/;
485       }
486
487       if ( $MEM{$domUuid}{memTotal} == 0 ) {
488         $meminfo = -1;
489         next;
490       } # If Total Mem == 0, then something went wrong (It's impossible to have 0KB of RAM). Reread ↙
              ↳ the samples.
491
492       $MEM{$domUuid}{memActuallyFree} = $MEM{$domUuid}{memUnused} + ↙
              ↳ $MEM{$domUuid}{memBuffers} + $MEM{$domUuid}{memCached};
493       $MEM{$domUuid}{memActuallyUsed} = $MEM{$domUuid}{memTotal} - ↙
              ↳ $MEM{$domUuid}{memActuallyFree};
494       $MEM{$domUuid}{swapActuallyFree} = $MEM{$domUuid}{swapFree} + ↙
              ↳ $MEM{$domUuid}{swapCached};
495       $MEM{$domUuid}{swapActuallyUsed} = $MEM{$domUuid}{swapTotal} - ↙
              ↳ $MEM{$domUuid}{swapActuallyFree};
496
497     }
498     $socket->close;
499
500     return \%MEM;
501
502   } else {
503
504     my $message = "Socket for $domName was not defined\n";
505     send_email_to_admins( $message, 'fatal' ) if ( !$IGNORE );
506     die $message;
507
508   }
509
510 }
511
512 sub delay_and_get_current_mem_5_samples_avg {
513
514   my $delay = $_[0];
515
516   my $delay_between_sampling = $delay / 5;
517
518   my %MEM;
519
520   foreach my $dom ( get_active_domains($virtualMachineManager) ) {
521     $MEM{ $dom->get_uuid_string }{name} = $dom->get_name;
522   }
```

```
523
524   for ( my $i = 0; $i < 5; $i++ ) {
525
526     my $loopStart = [ Time::HiRes::gettimeofday() ];
527
528     foreach my $domUuid ( keys %MEM ) {
529       my $RESULT = get_guest_memory( $domUuid, $MEM{$domUuid}{name} );
530       foreach my $key ( keys %{ $$RESULT{$domUuid} } ) {
531         $MEM{$domUuid}{$key} = $$RESULT{$domUuid}{$key};
532       }
533
534       $MEM{$domUuid}{CurrentMem5samplesAvg} += ( $MEM{$domUuid}{memActuallyUsed} + ↙
              ↳ $MEM{$domUuid}{swapActuallyUsed} );
535     }
536
537     my $timeNeededToRun = Time::HiRes::tv_interval($loopStart);
538     Time::HiRes::sleep( $delay_between_sampling - $timeNeededToRun ) if $timeNeededToRun < ↙
            ↳ $delay_between_sampling;
539   }
540
541   $MEM{$_}{CurrentMem5samplesAvg} = floor( $MEM{$_}{CurrentMem5samplesAvg} / 5 ) ↙
          ↳ foreach ( keys %MEM );
542
543   return \%MEM;
544
545 }
546
547 sub usage {
548
549   # prints the correct use of this script
550   say "Usage:";
551   say "-h Usage";
552   say "-v Verbose";
553   say "-d Debug";
554   say "-i Disable warning email notifications";
555
556 }
557
558 sub verbose {
559   say $_[0] if ($VERBOSE);
560 }
561
562 sub debug {
563   say $_[0] if ($DEBUG);
564 }
```

Listing A.2: loadsim.pl

```
1  #!/usr/bin/perl
2
3  # Vangelis Tasoulas <evanget@ifi.uio.no>
4
5  # Bayllocator -- Script to generate random CPU and Memory utilization.
6
7  # Needed packages
8  use lib '/usr/local/lib/bayllocator';
9  use lib 'lib';
10 use lib '../lib';
11 use strict 'vars';
12 use Getopt::Std;
13 use Math::BigInt;
14 use Time::HiRes qw(sleep usleep nanosleep);
15 use threads (
16   'yield',
17   'stack_size' => 64 * 4096,
18   'exit' => 'threads_only',
```

```
    'stringify'
);

# Global variables
my $VERBOSE = 0;
my $DEBUG = 0;

####################
# handle flags and arguments
# Example: c == "-c", c: == "-c argument"
my $opt_string = 'hvdm:c:M:';
getopts( "$opt_string", \my %opt ) or exit 1;

usage() && exit 0 if ( $opt{'h'} );

$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};

##################################
##################################
##################################
# Main program content #
##################################
##################################
##################################

my $max_memory_to_use = 600; # MB of Memory.
my $memThreads = 1; # Number of Threads to Share the $max_memory_to_use. Set to 0 if you do not need any memory usage generation.
my $cpuThreads = 1; # Number of Processing Threads to Open. Set to 0 if you do not need any cpu load generation.

$memThreads = $opt{'m'} if $opt{'m'};
$cpuThreads = $opt{'c'} if $opt{'c'};
$max_memory_to_use = $opt{'M'} if $opt{'M'};

debug("$memThreads concurrent memory threads will consume randomly up to $max_memory_to_use MB of RAM\n") if $memThreads > 0;
debug("$cpuThreads concurrent CPU threads will be spawned as well.\n") if $cpuThreads > 0;

while (1) {
  my %THREAD_HASH;

  for ( my $i = 1; $i <= $memThreads; $i++ ) {
    $THREAD_HASH{ 'mem' . $i } = threads->create( 'memory_load', ( $max_memory_to_use / $memThreads ) );
  }

  for ( my $i = 1; $i <= $cpuThreads; $i++ ) {
    $THREAD_HASH{ 'cpu' . $i } = threads->create('cpu_load');
  }

  for ( my $i = 1; $i <= $memThreads; $i++ ) {
    $THREAD_HASH{ 'mem' . $i }->join();
  }

  for ( my $i = 1; $i <= $cpuThreads; $i++ ) {
    $THREAD_HASH{ 'cpu' . $i }->join();
  }
}

# End opf Main program

##################################
##################################
##################################
```

```
81 # Helper routines #
82 ###################################
83 ###################################
84 ###################################
85
86 sub cpu_load {
87   my $x = 11000;
88   my $y = int( rand(13000) );
89   my $randomSleep = int( rand(3000000) );
90
91   Math::BigInt::bpow( $x, $y ); # Increase $x and $y to create more cpu load.
92                                 # Math::BigInt::bpow(11000, 15000);
93   usleep $randomSleep; # Sleep randomly up to $randomSleep microseconds.
94 }
95
96 sub memory_load {
97   my ($maxMemConsumption) = @_;
98
99   my $x = int( rand( $maxMemConsumption * 510000 ) );
100  my $randomSleep = int( rand(3000000) );
101
102  $_ = "x" x $x; # Repeat char "x" for $x times (this will consume lots of memory if $x is big enough!
103                 # Roughly1MB of memory will be consumed for every 510.000 $x.
104
105  usleep $randomSleep;
106 }
107
108 sub verbose {
109   print $_[0] if ($VERBOSE);
110 }
111
112 sub debug {
113   print $_[0] if ($DEBUG);
114 }
115
116 sub usage {
117   print "Usage:\n";
118   print "-h Usage.\n";
119   print "-v Verbose.\n";
120   print "-d Debug.\n";
121   print "-M Max memory (given in MB) to consume (default 600MB).\n";
122   print "-m Memory threads to open (All of the threads combined will consume a max of '-M' ↙
          ↪ memory).\n";
123   print "-c Cpu threads to open.\n";
124 }
```

Listing A.3: randomize-loadsim.sh

```
1  #!/bin/bash
2
3  while [ 1 ]; do
4
5    Delay=$(( RANDOM %= 1333 ))
6    MemThreads=$((RANDOM%10))
7    MemTotal=$((RANDOM%1000+400))
8
9    if [ $MemThreads -gt 0 ]; then
10     echo "/usr/bin/perl -e 'alarm shift @ARGV; exec @ARGV' $Delay /bin/loadsim.pl -m ↙
           ↪ $MemThreads -M $MemTotal -c 1"
11     /usr/bin/perl -e 'alarm shift @ARGV; exec @ARGV' $Delay /bin/loadsim.pl -m $MemThreads ↙
           ↪ -M $MemTotal -c 1
12   else
13     echo "sleep $Delay"
14     sleep $Delay
15   fi
16
```

```
done
```

Listing A.4: collect-data.pl

```
#!/usr/bin/perl

# Vangelis Tasoulas <evanget@ifi.uio.no>

# Bayllocator -- Collect Data from virtual machines script.

# Needed packages
use lib '/usr/local/lib/bayllocator';
use lib 'lib';
use lib '../lib';
use strict 'vars';
use Getopt::Std;
use feature 'say';
use Time::Local;
use Time::HiRes qw(sleep);
use POSIX qw(strftime);
use IO::Socket::UNIX qw( SOCK_STREAM );
use JSON;
use DBI;
use Clone qw( clone );
use Sys::Virt;
use MyLibVirtSubs;
use MyConfig;

# Global variables
my $VERBOSE = 0;
my $DEBUG = 0;

######################
# handle flags and arguments
# Example: c == "-c", c: == "-c argument"
my $opt_string = 'hvdf:s:i:';
getopts( "$opt_string", \my %opt ) or usage() and exit 1;

# print help message if -h is invoked
if ( $opt{'h'} ) {
   usage();
   exit 0;
}

$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};

my $dsn = "DBI:mysql:database=" . $config{db}{database} . ";host=" . $config{db}{host} . ";port=" . $config{db}{port};

my $virtualMachineManager = Sys::Virt->new( address => $config{virsh_host} );

##################################
##################################
##################################
# Main program content #
##################################
##################################
##################################

my %PREV_DATA = ();
my %CUR_DATA = ();

while (1) {
   my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or warn "Connection Error: $DBI::errstr\n";
```

```
61
62  if ( defined($db) ) {
63    my $loopStart = [ Time::HiRes::gettimeofday() ];
64
65    foreach my $dom ( get_active_domains($virtualMachineManager) ) {
66
67      my $domUuid = $dom->get_uuid_string;
68      my $domName = $dom->get_name;
69      my $domInfo = $dom->get_info;
70
71      my $socket_path = '/tmp/' . $domUuid . '.qemuga.sock';
72
73      my $socket = new IO::Socket::UNIX(
74        Type => SOCK_STREAM,
75        Peer => $socket_path
76      ) or warn "Can't connect to socket $socket_path: $!\n";
77      debug("$domName: Socket Opened\n");
78
79      if ( defined $socket ) {
80
81        my $mysqlTimestamp = -1;
82        my $uptime = -1;
83        my $netdev = -1;
84        my $meminfo = -1;
85        my $loadavg = -1;
86
87        my $memTotal = 0;
88        my $memUnused = 0;
89        my $memBuffers = 0;
90        my $memCached = 0;
91        my $swapTotal = 0;
92        my $swapFree = 0;
93        my $swapCached = 0; # Memory that once was swapped out, is swapped back in but still also ↙
              ↳ is in the swapfile (if memory is needed it doesn't need to be swapped out AGAIN ↙
              ↳ because it is already in the swapfile. This saves I/O)
94
95        my $memActuallyFree = 0;
96        my $memActuallyUsed = 0;
97        my $swapActuallyFree = 0;
98        my $swapActuallyUsed = 0;
99        my $memChange_from_last_sampling = 0;
100       my $swapChange_from_last_sampling = 0;
101
102       my $cpuCount = 0;
103
104       my $loadavg1 = 0.00;
105       my $loadavg5 = 0.00;
106       my $loadavg15 = 0.00;
107
108       my $uptimeSeconds = 0.00;
109       my $averageIdlingSecondsTotal = 0.00;
110       my $averageIdlingSeconds_from_last_sampling = 0.00;
111
112       my $rxBytesTotal = 0;
113       my $rxPacketsTotal = 0;
114       my $txBytesTotal = 0;
115       my $txPacketsTotal = 0;
116       my $totalBytes = 0;
117       my $totalPackets = 0;
118       my $rxBytes_from_last_sampling = 0;
119       my $rxPackets_from_last_sampling = 0;
120       my $txBytes_from_last_sampling = 0;
121       my $txPackets_from_last_sampling = 0;
122       my $totalBytes_from_last_sampling = 0;
123       my $totalPackets_from_last_sampling = 0;
124
```

```
125         # If any of qemuga_readfile() returns -1, retake samples.
126
127         while ( $uptime == -1 or $netdev == -1 or $meminfo == -1 or $loadavg == -1 ) {
128
129           debug("$domName: Entered While\n");
130
131           qemuga_syncguest($socket);
132
133           debug("$domName: Guest now in Sync\n");
134
135           $mysqlTimestamp = strftime( '%Y-%m-%d %H:%M:%S', localtime );
136           $uptime = qemuga_readfile( $socket, "/proc/uptime" );
137           $netdev = qemuga_readfile( $socket, "/proc/net/dev" );
138           $meminfo = qemuga_readfile( $socket, "/proc/meminfo" );
139           $loadavg = qemuga_readfile( $socket, "/proc/loadavg" );
140
141           debug("$domName: Data read from the guest\n");
142
143           # Get Memory Values
144           my @lines = split /\n/, $meminfo;
145           foreach my $line (@lines) {
146             $memTotal = $1 if $line =~ /^MemTotal:\s+(\d+)/;
147             $memUnused = $1 if $line =~ /^MemFree:\s+(\d+)/;
148             $memBuffers = $1 if $line =~ /^Buffers:\s+(\d+)/;
149             $memCached = $1 if $line =~ /^Cached:\s+(\d+)/;
150             $swapTotal = $1 if $line =~ /^SwapTotal:\s+(\d+)/;
151             $swapFree = $1 if $line =~ /^SwapFree:\s+(\d+)/;
152             $swapCached = $1 if $line =~ /^SwapCached:\s+(\d+)/;
153           }
154
155           if ( $memTotal == 0 ) {
156             $meminfo = -1;
157             next;
158           } # If Total Mem == 0, then something went wrong (It's impossible to have 0KB of RAM). ↙
                 ↳ Reread the samples.
159
160           $memActuallyFree = $memUnused + $memBuffers + $memCached;
161           $memActuallyUsed = $memTotal - $memActuallyFree;
162           $swapActuallyFree = $swapFree + $swapCached;
163           $swapActuallyUsed = $swapTotal - $swapActuallyFree;
164
165           $CUR_DATA{$domUuid}{'memActuallyUsed'} = $memActuallyUsed;
166           $CUR_DATA{$domUuid}{'swapActuallyUsed'} = $swapActuallyUsed;
167           if ( defined $PREV_DATA{$domUuid} ) {
168             $swapChange_from_last_sampling = $CUR_DATA{$domUuid}{'swapActuallyUsed'} - ↙
                     ↳ $PREV_DATA{$domUuid}{'swapActuallyUsed'};
169             $memChange_from_last_sampling = $CUR_DATA{$domUuid}{'memActuallyUsed'} - ↙
                     ↳ $PREV_DATA{$domUuid}{'memActuallyUsed'};
170           }
171
172           #Get Load Average, Uptime and Idling time.
173           $cpuCount = $$domInfo{'nrVirtCpu'};
174
175           $loadavg1 = 0.00;
176           $loadavg5 = 0.00;
177           $loadavg15 = 0.00;
178           if ( $loadavg =~ /^([0-9]*\.?[0-9]*)\s+([0-9]*\.?[0-9]*)\s+([0-9]*\.?[0-9]*)/ ) {
179             $loadavg1 = $1 / $cpuCount;
180             $loadavg5 = $2 / $cpuCount;
181             $loadavg15 = $3 / $cpuCount;
182           }
183
184           $uptimeSeconds = 0.00;
185           $averageIdlingSecondsTotal = 0.00;
186           if ( $uptime =~ /^([0-9]*\.?[0-9]*)\s+([0-9]*\.?[0-9]*)/ ) {
187             $uptimeSeconds = $1;
```

```
188             $averageIdlingSecondsTotal = $2 / $cpuCount;
189           }
190
191           if ( $uptimeSeconds == 0 ) {
192             $uptime = -1;
193             next;
194           } # If uptime == 0, then something went wrong (It's impossible to have 0 uptime if the ↙
                  ↳ machine is up). Reread the samples.
195
196           $CUR_DATA{$domUuid}{'averageIdlingSecondsTotal'} = $averageIdlingSecondsTotal;
197           $averageIdlingSeconds_from_last_sampling = $averageIdlingSecondsTotal;
198           if ( defined $PREV_DATA{$domUuid} ) {
199             $averageIdlingSeconds_from_last_sampling = ↙
                    ↳ $CUR_DATA{$domUuid}{'averageIdlingSecondsTotal'} - ↙
                    ↳ $PREV_DATA{$domUuid}{'averageIdlingSecondsTotal'};
200           }
201
202           $rxBytesTotal = 0;
203           $rxPacketsTotal = 0;
204           $txBytesTotal = 0;
205           $txPacketsTotal = 0;
206           @lines = split /\n/, $netdev;
207           foreach my $line (@lines) {
208             if ( $line =~ /^\s+eth\d+:\s+(\d+)\s+(\d+)(\s+\d+){6}\s+(\d+)\s+(\d+)/ ) {
209               $rxBytesTotal += $1;
210               $rxPacketsTotal += $2;
211               $txBytesTotal += $4;
212               $txPacketsTotal += $5;
213             }
214           }
215           $totalBytes = $rxBytesTotal + $txBytesTotal;
216           $totalPackets = $rxPacketsTotal + $txPacketsTotal;
217
218           $CUR_DATA{$domUuid}{'rxBytesTotal'} = $rxBytesTotal;
219           $CUR_DATA{$domUuid}{'rxPacketsTotal'} = $rxPacketsTotal;
220           $CUR_DATA{$domUuid}{'txBytesTotal'} = $txBytesTotal;
221           $CUR_DATA{$domUuid}{'txPacketsTotal'} = $txPacketsTotal;
222           $rxBytes_from_last_sampling = $rxBytesTotal;
223           $rxPackets_from_last_sampling = $rxPacketsTotal;
224           $txBytes_from_last_sampling = $txBytesTotal;
225           $txPackets_from_last_sampling = $txPacketsTotal;
226           $totalBytes_from_last_sampling = $totalBytes;
227           $totalPackets_from_last_sampling = $totalPackets;
228
229           if ( defined $PREV_DATA{$domUuid} ) {
230             $rxBytes_from_last_sampling = $CUR_DATA{$domUuid}{'rxBytesTotal'} - ↙
                    ↳ $PREV_DATA{$domUuid}{'rxBytesTotal'};
231             $rxPackets_from_last_sampling = $CUR_DATA{$domUuid}{'rxPacketsTotal'} - ↙
                    ↳ $PREV_DATA{$domUuid}{'rxPacketsTotal'};
232             $txBytes_from_last_sampling = $CUR_DATA{$domUuid}{'txBytesTotal'} - ↙
                    ↳ $PREV_DATA{$domUuid}{'txBytesTotal'};
233             $txPackets_from_last_sampling = $CUR_DATA{$domUuid}{'txPacketsTotal'} - ↙
                    ↳ $PREV_DATA{$domUuid}{'txPacketsTotal'};
234             $totalBytes_from_last_sampling = $CUR_DATA{$domUuid}{'totalBytes'} - ↙
                    ↳ $PREV_DATA{$domUuid}{'totalBytes'};
235             $totalPackets_from_last_sampling = $CUR_DATA{$domUuid}{'totalPackets'} - ↙
                    ↳ $PREV_DATA{$domUuid}{'totalPackets'};
236           }
237         }
238
239         verbose("Host $domName \n");
240         verbose("----------------------------------------\n");
241         verbose("Data Retrieved at $mysqlTimestamp <-\n");
242         verbose("----------------------------------------\n");
243         verbose("memTotal = $memTotal <-\n");
244         verbose("memUnused = $memUnused\n");
```

```
245             verbose("memBuffers = $memBuffers\n");
246             verbose("memCached = $memCached\n");
247             verbose("memActuallyFree = $memActuallyFree\n");
248             verbose("memActuallyUsed = $memActuallyUsed <-\n");
249             verbose( "memTotalBySum = " . ( $memActuallyFree + $memActuallyUsed ) . "\n" );
250             verbose("swapTotal = $swapTotal <-\n");
251             verbose("swapActuallyFree = $swapActuallyFree\n");
252             verbose("swapActuallyUsed = $swapActuallyUsed <-\n");
253             verbose("memChange_from_last_sampling = $memChange_from_last_sampling\n");
254             verbose("swapChange_from_last_sampling = $swapChange_from_last_sampling\n");
255             verbose("----------Load Average and Uptime----------\n");
256             verbose("cpuCount = $cpuCount <-\n");
257             verbose("loadavg1 = $loadavg1 <-\n");
258             verbose("loadavg5 = $loadavg5 <-\n");
259             verbose("loadavg15 = $loadavg15 <-\n");
260             verbose("uptimeSeconds = $uptimeSeconds <-\n");
261             verbose("averageIdlingSecondsTotal = $averageIdlingSecondsTotal <-\n");
262             verbose("averageIdlingSeconds_from_last_sampling = ↙
                    ↳ $averageIdlingSeconds_from_last_sampling\n");
263             verbose("----------Network----------\n");
264             verbose("rxBytesTotal = $rxBytesTotal <-\n");
265             verbose("rxPacketsTotal = $rxPacketsTotal <-\n");
266             verbose("txBytesTotal = $txBytesTotal <-\n");
267             verbose("txPacketsTotal = $txPacketsTotal <-\n");
268             verbose("totalBytes = $totalBytes\n");
269             verbose("totalPackets = $totalPackets\n");
270             verbose("rxBytes_from_last_sampling = $rxBytes_from_last_sampling\n");
271             verbose("rxPackets_from_last_sampling = $rxPackets_from_last_sampling\n");
272             verbose("txBytes_from_last_sampling = $txBytes_from_last_sampling\n");
273             verbose("txPackets_from_last_sampling = $txPackets_from_last_sampling\n");
274             verbose("totalBytes_from_last_sampling = $totalBytes_from_last_sampling\n");
275             verbose("totalPackets_from_last_sampling = $totalPackets_from_last_sampling\n");
276             verbose("----------------------------------------\n\n");
277
278             store_data_to_db( $db, $domUuid, $mysqlTimestamp, $memTotal, $memActuallyUsed, ↙
                    ↳ $swapTotal, $swapActuallyUsed, $cpuCount, $loadavg1, $loadavg5, $loadavg15, ↙
                    ↳ $uptimeSeconds, $averageIdlingSecondsTotal, $rxBytesTotal, $rxPacketsTotal, ↙
                    ↳ $txBytesTotal, $txPacketsTotal ) if ( defined($db) );
279             debug("$domName: Data Stored to db\n");
280
281             $socket->close;
282             debug("$domName: Socket Closed\n");
283
284           } else {
285
286             next;
287
288           }
289         }
290
291         %PREV_DATA = %{ clone( \%CUR_DATA ) };
292
293         my $timeNeededToCollectData = Time::HiRes::tv_interval($loopStart);
294         verbose("Elapsed time: $timeNeededToCollectData seconds!\n\n");
295
296         # Disconnect from the database.
297         $db->disconnect();
298
299         # Sleep until the next $config{data_collection_interval} comes.
300         Time::HiRes::sleep $config{data_collection_interval} - $timeNeededToCollectData if ↙
                ↳ $timeNeededToCollectData < $config{data_collection_interval};
301
302       }
303     }
304
305   # End opf Main program
```

```
306
307 ################################
308 ################################
309 ################################
310 # Helper routines #
311 ################################
312 ################################
313 ################################
314
315 sub store_data_to_db {
316   my $db = shift @_;
317   my ( $domUuid, $date, $memTotal, $memActuallyUsed, $swapTotal, $swapActuallyUsed, ↙
          ↳ $cpuCount, $loadavg1, $loadavg5, $loadavg15, $uptimeSeconds, ↙
          ↳ $averageIdlingSecondsTotal, $rxBytesTotal, $txBytesTotal, $rxPacketsTotal, $txPacketsTotal ) ↙
          ↳ = @_;
318
319   #<<< do not let perltidy touch this
320   my $sql = "INSERT INTO '" . $config{db}{data_table} . "' (
321   'host_id', 'date',
322   'mem_total', 'mem_used',
323   'swap_total', 'swap_used',
324   'CPU_count', 'load_avg_1',
325   'load_avg_5', 'load_avg_15',
326   'uptime', 'cpuIdlingTime',
327   'rxBytesTotal', 'txBytesTotal',
328   'rxPacketsTotal', 'txPacketsTotal'
329   ) SELECT " . $config{db}{hosts_table} . ".id, '$date',
330   $memTotal, $memActuallyUsed,
331   $swapTotal, $swapActuallyUsed,
332   $cpuCount, $loadavg1,
333   $loadavg5, $loadavg15,
334   $uptimeSeconds, $averageIdlingSecondsTotal,
335   $rxBytesTotal, $txBytesTotal,
336   $rxPacketsTotal, $txPacketsTotal
337   FROM '" . $config{db}{hosts_table} . "'
338   WHERE '" . $config{db}{hosts_table} . "'.'uuid' = '$domUuid'";
339   #>>>
340   $db->do($sql) or warn "SQL Error: $DBI::errstr\n";
341 }
342
343 sub usage {
344
345   # prints the correct use of this script
346   print "Usage:\n";
347   print "-h Usage\n";
348   print "-v Verbose\n";
349   print "-d Debug\n";
350   print "-f filename Use this to update the database from a file. You can't mix this with another ↙
          ↳ option.\n";
351   print "-s starttime Specify the start time to search in this format \"YYYY-MM-DD HH:MM:SS\"\n";
352   print "-i Interval ahead \"-s\" value to return results. This is an integer value representing hours. ↙
          ↳ Default value is 24 hours.\n";
353 }
354
355 sub verbose {
356   print $_[0] if ($VERBOSE);
357 }
358
359 sub debug {
360   print $_[0] if ($DEBUG);
361 }
```

Listing A.5: data-smoothing.pl

```
1 #!/usr/bin/perl
2
```

```
3  # Vangelis Tasoulas <evanget@ifi.uio.no>
4
5  # Bayllocator -- Database Data Smoothing script.
6
7  # Needed packages
8  use lib '/usr/local/lib/bayllocator';
9  use lib 'lib';
10 use lib '../lib';
11 use strict 'vars';
12 use Getopt::Std;
13 use feature 'say';
14 use POSIX qw(strftime ceil floor);
15 use DBI;
16 use List::Util qw(sum);
17 use MyConfig;
18 use MyDBSubs;
19 use MyTimeSubs;
20
21 my $DEBUG = 0;
22 my $VERBOSE = 0;
23 my $UPDATEFROMSTART = 0;
24 my $FROMDATE = 0;
25 ####################
26 # handle flags and arguments
27 # Example: c == "-c", c: == "-c argument"
28 my $opt_string = 'dvhi:sD:';
29 getopts( "$opt_string", \my %opt ) or usage() and exit 1;
30
31 # print help message if -h is invoked
32 if ( $opt{'h'} ) {
33   usage();
34   exit 0;
35 }
36
37 $DEBUG = 1 if $opt{'d'};
38 $VERBOSE = 1 if $opt{'v'};
39 $UPDATEFROMSTART = 1 if $opt{'s'};
40 $FROMDATE = $opt{'D'} if $opt{'D'};
41
42 my $dsn = "DBI:mysql:database=" . $config{db}{database} . ";host=" . $config{db}{host} . ";port=" . ↙
       ↳ $config{db}{port};
43 my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or warn "Connection Error: ↙
       ↳ $DBI::errstr\n";
44
45 # Make SURE if you change the interval, that the table $config{db}{rolling_avg_table}
46 # contains three columns with the names:
47 # mem_avg_used_$interval
48 # mem_avg_total_$interval
49 # mem_used_rate_$interval
50 # If you choose $interval to be 15, then you should have
51 # mem_avg_used_15
52 # mem_avg_total_15
53 # mem_used_rate_15
54 my $interval = $config{averages_min};
55
56 $interval = $opt{'i'} if $opt{'i'};
57
58 ##################################
59 ##################################
60 ##################################
61 # Main program content #
62 ##################################
63 ##################################
64 ##################################
65
66 if ( defined($db) ) {
```

```
67
68    if ($UPDATEFROMSTART) {
69
70      update_db_from_datatable(0);
71
72    } elsif ( $FROMDATE != 0 ) {
73
74      if ( validate_datetime($FROMDATE) ) {
75        #<<< do not let perltidy touch this
76        my $data_id = $db->selectrow_array("SELECT `data_id`
77                                            FROM `$config{db}{rolling_avg_table}`
78                                            WHERE `date` >= DATE_SUB('$FROMDATE', INTERVAL ↙
                                                  ↳ $interval MINUTE)
79                                            ORDER BY `id`
80                                            LIMIT 1");
81        #>>>
82
83        update_db_from_datatable($data_id);
84      }
85
86    } else {
87
88      # First get the date from the last line of the table $config{db}{rolling_avg_table}
89      #<<< do not let perltidy touch this
90      my ( $last_stored_avg_id, $last_stored_date )= $db->selectrow_array("SELECT `id`, `date`
91                                                                          FROM ↙
                                                                                ↳ `$config{db}{rolling_avg_table}`
92                                                                          ORDER BY `id`
93                                                                          DESC LIMIT 1");
94      #>>>
95
96      # If last_stored_id is set (defined) then update the table
97      # If not, it means that the table is empty so go to
98      # the else statement and start building the table from scratch.
99      if ( defined($last_stored_avg_id) ) {
100
101       # The last rows contain incomplete averages since there are no samples following.
102       # When we add new data, find these rows $interval minutes back to improve their
103       # averages with the newly added data.
104       #<<< do not let perltidy touch this
105       my $data_id = $db->selectrow_array("SELECT `data_id`
106                                           FROM `$config{db}{rolling_avg_table}`
107                                           WHERE `date` >= DATE_SUB('$last_stored_date', INTERVAL ↙
                                                  ↳ $interval MINUTE)
108                                           ORDER BY `id`
109                                           LIMIT 1");
110       #>>>
111
112       # Then iterate in the data table from the $data_id row until the last row
113       update_db_from_datatable($data_id);
114
115     } else {
116
117       # This else will run only if the $config{db}{rolling_avg_table} table is empty
118       # and it will start filling it up by starting at row 0 on table $config{db}{data_table}
119       update_db_from_datatable(0);
120     }
121
122   }
123
124   $db->disconnect();
125
126 } else {
127
128   warn "Could not connect to the database.";
129
```

```
130 }
131
132 # End opf Main program
133
134 ##################################
135 ##################################
136 ##################################
137 # Helper routines #
138 ##################################
139 ##################################
140 ##################################
141
142 sub update_db_from_datatable {
143
144   my $start_row_id = $_[0];
145
146   my $start_row = get_row_number_of_table_by_id( $config{db}{data_table}, $start_row_id ) - 1;
147
148   # If the get_row_number_of_datatable_by_id($start_row_id) returns 0, this means
149   # that the $start_row_id could not be found in the data table, so set the $start_row
150   # to 0 and iterate from the beginning of the data table.
151   $start_row = 0 if $start_row == -1;
152
153   # Get the max number of rows of the data table
154   my $max_rows = get_max_rows_of_table( $config{db}{data_table} );
155
156   # Then iterate through the whole $config{db}{data_table} table to create the rolling averages.
157   for ( my $i = $start_row; $i < $max_rows; $i++ ) {
158
159     # Run the smoothdata_query to create the rolling averages
160     # for the next $interval minutes given the current row $i.
161     my $vars = smoothdata_query( $i, $interval );
162
163     # Store the results to the intermediate table $config{db}{rolling_avg_table}
164     # You could create another subroutine to save them into a file as well, or
165     # redirect the verbose output
166     store_or_update_db( $$vars{'id'}, $$vars{'host_id'}, $$vars{'date'}, $$vars{ 'mem_avg_total_' . ↙
          ↪ $interval }, $$vars{ 'mem_avg_used_' . $interval }, $$vars{ 'mem_used_rate_' . $interval } );
167
168     # Show the output to the user if they choose -v command line argument
169     # Print labels if it's the first line.
170     verbose("date,host_id,mem_avg_total_$interval,mem_avg_used_$interval,mem_used_rate_$interval") ↙
          ↪ if ( $i == 0 );
171     verbose( $$vars{'date'} . "," . $$vars{'host_id'} . "," . $$vars{ 'mem_avg_total_' . $interval } . "," . ↙
          ↪ $$vars{ 'mem_avg_used_' . $interval } . "," . $$vars{ 'mem_used_rate_' . $interval } );
172   }
173 }
174
175 # smoothdata_query accepts two arguments.
176 # $count: The starting row
177 # $interval: The number of minutes to summarize for the average following the row $count
178 # Returns
179 # mem_avg_used_$interval
180 # mem_avg_total_$interval
181 # mem_used_rate_$interval
182 sub smoothdata_query {
183   my $count = $_[0];
184   my $interval = $_[1]; # Smoothing window in minutes
185
186   my %smoothing;
187
188   # First select only one row (the $count which is the starting row)
189   # to read the starting $date, host_id and mem_total from.
190   #<<< do not let perltidy touch this
191   my ( $id, $date, $host_id, $mem_total ) = $db->selectrow_array("SELECT 'id', 'date', 'host_id', ↙
          ↪ 'mem_total'
```

```
                                                    FROM `$config{db}{data_table}`
                                                    ORDER BY `id`
                                                    LIMIT $count, 1;");
    #>>>

    my %returnvars;
    $returnvars{'id'} = $id;
    $returnvars{'date'} = $date;
    $returnvars{'host_id'} = $host_id;

    # Then select $interval minutes in front of the $count row selected
    # before to create an $interval minutes average.
    #<<< do not let perltidy touch this
    my $sql = "SELECT `mem_used`, `swap_used`, `mem_total`
    FROM `$config{db}{data_table}`
    WHERE `host_id` = $host_id
    AND (
    `date` BETWEEN '$date'
    AND DATE_ADD('$date', INTERVAL $interval MINUTE)
    );";
    #>>>

    my $sth = $db->prepare($sql);
    $sth->execute();
    while ( my $ref = $sth->fetchrow_hashref() ) {
        push( @{ $smoothing{mem_used} }, $ref->{'mem_used'} + $ref->{'swap_used'} );
        push( @{ $smoothing{mem_total} }, $ref->{'mem_total'} );
    }
    $sth->finish();
    my $avg_used = ceil( avg_array( @{ $smoothing{mem_used} } ) );
    my $avg_total = ceil( avg_array( @{ $smoothing{mem_total} } ) );
    $returnvars{ 'mem_avg_used_' . $interval } = $avg_used;
    $returnvars{ 'mem_avg_total_' . $interval } = $avg_total;
    $returnvars{ 'mem_used_rate_' . $interval } = $avg_used / $avg_total;

    return \%returnvars;
}

sub store_or_update_db {
    my ( $id, $host_id, $date, $mem_avg_total, $mem_avg_used, $mem_used_rate ) = @_;

    #<<< do not let perltidy touch this
    my $sql = "INSERT INTO `$config{db}{rolling_avg_table}` (
    `data_id`,
    `data_host_id`,
    `date`,
    `mem_avg_used_$interval`,
    `mem_avg_total_$interval`,
    `mem_used_rate_$interval`
    ) VALUES (
    $id,
    $host_id,
    '$date',
    $mem_avg_used,
    $mem_avg_total,
    $mem_used_rate
    ) ON DUPLICATE KEY UPDATE
    `mem_avg_used_$interval` = $mem_avg_used,
    `mem_avg_total_$interval` = $mem_avg_total,
    `mem_used_rate_$interval` = $mem_used_rate;";
    #>>>
    my $sth = $db->prepare($sql);
    $sth->execute();
    $sth->finish();
}

```

```
sub avg_array {
  return sum(@_) / @_;
}

sub usage {
  say "Usage:";
  say "-h Usage";
  say "-v Verbose";
  say "-d Debug";
  say "-i Define average interval in minutes (default 15)";
  say " Make sure that you have the need columns in the average table";
  say " Check the comments in the script for more information";
  say "-s Force the building of averages from the 1st entry exists in the data table";
  say "-D Define the datetime where you want to start building averages from";
}

sub debug {
  say $_[0] if $DEBUG;
}

sub verbose {
  say $_[0] if $VERBOSE;
}
```

Listing A.6: data-to-bn-states.pl

```
#!/usr/bin/perl

# Vangelis Tasoulas <evanget@ifi.uio.no>

# Bayllocator -- Data Conversion to discrete states.
# This script is responsible to create the files the
# will read and pass to the R script.

# Needed packages
use lib '/usr/local/lib/bayllocator';
use lib 'lib';
use lib '../lib';
use strict 'vars';
use Getopt::Std;
use feature 'say';
use POSIX qw(floor ceil);
use MyTimeSubs;
use DBI;
use Data::Dumper;
use List::MoreUtils qw(uniq);
use sigtrap
  'handler' => \&cleanAndExit,
  'INT', 'ABRT', 'QUIT', 'TERM';
use MyConfig;
use MyDBSubs;
use MyBayesianPredictor;

my $DEBUG = 0;
my $VERBOSE = 0;
my $OUTPUT_TO_FILE = 0;
my $SKIP_FILEPATH = 0;
my $START_DATE = -1;
my $END_DATE = -1;
my $RShiftColumn_BIN = "ShiftColumn.R";

my $opt_string = 'dvhH:osS:E:';
getopts( "$opt_string", \my %opt ) or usage() and exit 1;

# print help message if -h is invoked
if ( $opt{'h'} ) {
```

```
41    usage();
42    exit 0;
43  }
44
45  $DEBUG = 1 if $opt{'d'};
46  $VERBOSE = 1 if $opt{'v'};
47  $OUTPUT_TO_FILE = 1 if $opt{'o'};
48  $SKIP_FILEPATH = 1 if $opt{'s'};
49
50  if ( $opt{'S'} ) {
51    $START_DATE = $opt{'S'};
52    $START_DATE = timestamp_to_datetime($START_DATE) if ( isnumeric($START_DATE) );
53  }
54
55  if ( $opt{'E'} ) {
56    $END_DATE = $opt{'E'};
57    $END_DATE = timestamp_to_datetime($END_DATE) if ( isnumeric($END_DATE) );
58  }
59
60  my $dsn = "DBI:mysql:database=" . $config{db}{database} . ";host=" . $config{db}{host} . ";port=" . ↙
        ↳ $config{db}{port};
61  my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or warn "Connection Error: ↙
        ↳ $DBI::errstr\n";
62
63  ####################
64
65  my $interval = 1; # In minutes. $interval here is used to fill
66                    # the missing timestamps (if we have missed
67                    # some data during the data collection.
68
69  ##################################
70  ##################################
71  ##################################
72  # Main program content #
73  ##################################
74  ##################################
75  ##################################
76
77  if ( defined($db) ) {
78
79    # If the parameter H is passed with a valid guest name
80    # then return results only for this guest.
81    # Otherwise return results for all of guests with data
82    # in the database
83    if ( $opt{'H'} ) {
84
85      my $guestname = $opt{'H'};
86      my $host_id = get_hostid_by_name($guestname);
87      convert_data_from_db($host_id) if ( $host_id != -1 );
88
89    } else {
90
91      my @host_ids = get_host_ids_from_db();
92
93      foreach my $host_id (@host_ids) {
94        convert_data_from_db($host_id);
95      }
96
97    }
98
99  }
100
101 # End opf Main program
102
103 ##################################
104 ##################################
```

```
105 ##################################
106 # Helper routines #
107 ##################################
108 ##################################
109 ##################################
110
111 sub convert_data_from_db {
112
113   my $hostid = $_[0];
114
115   my $query_add;
116   if ( $START_DATE != -1 and $END_DATE != -1 ) {
117     $query_add = " AND (date BETWEEN '$START_DATE' and '$END_DATE')";
118   } elsif ( $START_DATE != -1 and $END_DATE == -1 ) {
119     $query_add = " AND date >= '$START_DATE'";
120   } elsif ( $START_DATE == -1 and $END_DATE != -1 ) {
121     $query_add = " AND date <= '$END_DATE'";
122   }
123
124   my $max_rows_of_avg_table = 0;
125
126   $max_rows_of_avg_table = get_max_rows_of_table_per_host( $config{db}{rolling_avg_table}, $hostid, ↙
        ↳ $query_add );
127
128   if ( $max_rows_of_avg_table > 0 ) {
129
130     my $filename = get_hostname_by_id($hostid) . $config{bayesian_state_files_ext};
131     $filename = $config{bayesian_state_files_path} . $filename if ( !$SKIP_FILEPATH );
132     open FILE, ">" . $filename or die $! if ($OUTPUT_TO_FILE);
133
134     # Read the first row just to get the earliest date.
135     #<<< do not let perltidy touch this
136     my ( $prevdate, $mem_avg_used ) = $db->selectrow_array("SELECT `date`, ↙
          ↳ `mem_avg_used_$config{averages_min}`
137                                                             FROM `$config{db}{rolling_avg_table}`
138                                                             WHERE `data_host_id` = $hostid ↙
                                                                    ↳ $query_add
139                                                             ORDER BY `id`
140                                                             ASC
141                                                             LIMIT 1");
142     #>>>
143
144     # Print the headers
145     my $outputline = join( ",", @{ $bayesian_states{headers} } );
146     verbose($outputline);
147     say FILE $outputline if ($OUTPUT_TO_FILE);
148
149     # Add all of the available states to the output file to avoid NaN results from QueryNet.R
150     my %REVERSE_HASH = reverse %{ $bayesian_states{memory_util} };
151     foreach my $value ( uniq( values %REVERSE_HASH ) ) {
152       $outputline = create_output_line( $bayesian_states{missing_values}, ( $value * 100 * 1024 ) );
153       verbose($outputline);
154       say FILE $outputline if ($OUTPUT_TO_FILE);
155     }
156
157     # Assign it to the previous $previous_datetime and then get the whole table
158     # starting from the second result to iterate through.
159     my $previous_datetime = timestamp_to_the_nearest_whole_minute( ↙
          ↳ datetime_to_timestamp($prevdate) );
160     $outputline = create_output_line( $previous_datetime, $mem_avg_used );
161     verbose($outputline);
162     say FILE $outputline if ($OUTPUT_TO_FILE);
163
164     #<<< do not let perltidy touch this
165     my $sql = "SELECT `date`, `mem_avg_used_$config{averages_min}`
166     FROM `$config{db}{rolling_avg_table}`
```

```
167     WHERE ‘data_host_id‘ = $hostid $query_add
168     ORDER BY ‘id‘
169     ASC
170     LIMIT 1, $max_rows_of_avg_table”;
171     #>>>
172     my $sth = $db->prepare($sql);
173     $sth->execute() or die ”SQL Error: $DBI::errstr\n”;
174
175     while ( my $ref = $sth->fetchrow_hashref() ) {
176       my $current_datetime = timestamp_to_the_nearest_whole_minute( datetime_to_timestamp( ↙
              ↳ $ref->{’date’} ) );
177
178       # The following line ensures that we will avoid duplicate output lines
179       # due to timestamp_to_the_nearest_whole_minute. We might lose some data
180       # but this is rare and negligible in a huge table like the one we have
181       # meaning our final probabilities will not be affected.
182       next if ( $current_datetime == $previous_datetime );
183
184       # Add missing lines every 1 minute.
185       my @missing_lines_time = get_missing_timestamps( $previous_datetime, $current_datetime, 60 );
186       if ( @missing_lines_time[0] != -1 ) {
187         foreach my $missing_timestamp (@missing_lines_time) {
188           $outputline = create_output_line($missing_timestamp);
189           verbose($outputline);
190           say FILE $outputline if ($OUTPUT_TO_FILE);
191         }
192       }
193
194       $outputline = create_output_line( $current_datetime, $ref->{’mem_avg_used_’ . ↙
              ↳ $config{averages_min}} );
195       verbose($outputline);
196       say FILE $outputline if ($OUTPUT_TO_FILE);
197
198       $previous_datetime = $current_datetime;
199     }
200     $sth->finish();
201
202     close(FILE) if ($OUTPUT_TO_FILE);
203
204     my $shift_lines = abs( floor( $config{predict_future} / $config{data_collection_interval} ) );
205     $shift_lines = 1 if ( $shift_lines == 0 );
206     if ($OUTPUT_TO_FILE) {
207       my $command = $RShiftColumn_BIN . ” ’$filename’ ’CMU’ ” . $shift_lines . ” ’” . ↙
              ↳ $bayesian_states{missing_values} . ”’”;
208       system($command);
209       #$command = $RShiftColumn_BIN . ” ’$filename’ ’PMU’ ” . ( $shift_lines * 2 ) . ” ’” . ↙
              ↳ $bayesian_states{missing_values} . ”’”;
210       #system($command);
211     }
212   }
213
214 }
215
216 # This function adds the missing data timestamps
217 # Inputs
218 # $previous_line <- a valid timestamp
219 # $current_line <- a valid timestamp
220 # $add_every_Xseconds <- Define the number of seconds in between
221 sub get_missing_timestamps {
222
223   my ( $previous_line, $current_line, $add_every_Xseconds ) = @_;
224
225   validate_timestamp($previous_line);
226   validate_timestamp($current_line);
227
228   my @missing_lines;
```

```
229
230     my $diff = timestamp_diff_in_seconds( $previous_line, $current_line );
231
232     if ( $diff > $add_every_Xseconds ) {
233       for ( my $i = $add_every_Xseconds; $i < $diff; $i += $add_every_Xseconds ) {
234         push( @missing_lines, $previous_line += $add_every_Xseconds );
235       }
236       return @missing_lines;
237     }
238     return -1;
239
240 }
241
242 # http://www.linux-bsd-central.com/index.php/content/view/41/35/
243 # Remove temp files in case you received a kill signal
244 sub cleanAndExit {
245
246     say "\nCaught a kill signal, cleaning up temp files and exiting";
247
248     #unlink($output_filename);
249     exit 1;
250
251 }
252
253 sub usage {
254
255     say "Usage:";
256     say "-h Usage";
257     say "-v Verbose";
258     say "-d Debug";
259     say "-H <guest name>";
260     say " If a guest name is defined, only data for this guest will be parsed from the db";
261     say "-o Create output file";
262     say " If -o is used, an output filename \"<GuestName>.dat\" will be created for each host.";
263     say " This file can be parsed directly using the R scripts to calculate the probabilities.";
264     say "-s Save to the local directory instead of the path defined by the config files.";
265     say "-S Start Date (can be a datetime or timestamp)";
266     say "-E End Date (can be a datetime or timestamp)";
267
268 }
269
270 sub debug {
271
272     say $_[0] if $DEBUG;
273
274 }
275
276 sub verbose {
277
278     say $_[0] if $VERBOSE;
279
280 }
```

Listing A.7: data-from-file-to-bn-states.pl

```
1  #!/usr/bin/perl
2
3  # Vangelis Tasoulas <evanget@ifi.uio.no>
4
5  # Bayllocator -- Data Conversion to discrete states.
6  # This script is responsible to create the files the
7  # will read and pass to the R script.
8
9  # Needed packages
10 use lib '/usr/local/lib/bayllocator';
11 use lib 'lib';
```

```
12 use lib '../lib';
13 use strict 'vars';
14 use Getopt::Std;
15 use feature 'say';
16 use POSIX qw(floor ceil);
17 use MyTimeSubs;
18 use DBI;
19 use Data::Dumper;
20 use List::MoreUtils qw(uniq);
21 use sigtrap
22   'handler' => \&cleanAndExit,
23   'INT', 'ABRT', 'QUIT', 'TERM';
24 use MyConfig;
25 use MyDBSubs;
26 use MyBayesianPredictor;
27
28 my $DEBUG = 0;
29 my $VERBOSE = 0;
30 my $OUTPUT_TO_FILE = 0;
31 my $START_DATE = -1;
32 my $END_DATE = -1;
33 my $RShiftColumn_BIN = "ShiftColumn.R";
34 my $OUTPUT_FILENAME;
35 my $INPUT_FILE;
36
37 my $opt_string = 'dvhS:E:f:o:';
38 getopts( "$opt_string", \my %opt ) or usage() and exit 1;
39
40 # print help message if -h is invoked
41 if ( $opt{'h'} ) {
42   usage();
43   exit 0;
44 }
45
46 if ( !$opt{'f'} ) {
47   say "An input file to read the data is required.";
48   say "The input file has to be collected or to be in the same format as the files generated by the ↙
         ↳ system-data-collection.sh script.";
49   usage();
50   exit 1;
51 }
52
53 $DEBUG = 1 if $opt{'d'};
54 $VERBOSE = 1 if $opt{'v'};
55 $OUTPUT_TO_FILE = 1 if $opt{'o'};
56 $OUTPUT_FILENAME = $opt{'o'} if $opt{'o'};
57 $INPUT_FILE = $opt{'f'};
58
59 if ( $opt{'S'} ) {
60   $START_DATE = $opt{'S'};
61   $START_DATE = datetime_to_timestamp($START_DATE) if ( !isnumeric($START_DATE) );
62 }
63
64 if ( $opt{'E'} ) {
65   $END_DATE = $opt{'E'};
66   $END_DATE = datetime_to_timestamp($END_DATE) if ( !isnumeric($END_DATE) );
67 }
68
69 ####################
70
71 my $interval = 1; # In minutes. $interval here is used to fill
72                   # the missing timestamps (if we have missed
73                   # some data during the data collection.
74
75 ##################################
76 ##################################
```

```
77  ##################################
78  # Main program content #
79  ##################################
80  ##################################
81  ##################################
82
83  convert_data_from_file();
84
85  # End opf Main program
86
87  ##################################
88  ##################################
89  ##################################
90  # Helper routines #
91  ##################################
92  ##################################
93  ##################################
94
95  sub convert_data_from_file {
96
97    open INPUT_FILE, $INPUT_FILE or die $!;
98    open OUTPUT_FILE, ">" . $OUTPUT_FILENAME or die $! if ($OUTPUT_TO_FILE);
99
100   # Read the first row just to get the headers.
101   my $line = <INPUT_FILE>;
102   chomp($line);
103   my @input_headers = split( /,/, $line );
104   my %INPUT_HEADERS;
105
106   #$INPUT_HEADERS{$_}{}
107   for ( my $i = 0; $i < ( scalar @input_headers ); $i++ ) {
108     $INPUT_HEADERS{ @input_headers[$i] } = $i;
109   }
110
111   # Print the headers
112   my $outputline = join( ",", @{ $bayesian_states{headers} } );
113   verbose($outputline);
114   say OUTPUT_FILE $outputline if ($OUTPUT_TO_FILE);
115
116   # Add all of the available states to the output file to avoid NaN results from QueryNet.R
117   my %REVERSE_HASH = reverse %{ $bayesian_states{memory_util} };
118   foreach my $value ( uniq( values %REVERSE_HASH ) ) {
119     $outputline = create_output_line( $bayesian_states{missing_values}, ( $value * 100 * 1024 ) );
120     verbose($outputline);
121     say OUTPUT_FILE $outputline if ($OUTPUT_TO_FILE);
122   }
123
124   my $prevdate;
125   my @columns;
126   my $current_mem_used;
127   my $previous_datetime;
128   my $current_datetime;
129
130   while ( my $line = <INPUT_FILE> ) {
131     chomp($line);
132     @columns = split( /,/, $line );
133     $prevdate = @columns[ $INPUT_HEADERS{timestamp} ];
134     $current_mem_used = @columns[ $INPUT_HEADERS{memActuallyUsed} ] + @columns[ ↙
          ↳ $INPUT_HEADERS{swapActuallyUsed} ];
135     last if check_if_in_given_range($prevdate);
136   }
137
138   # Assign it to the previous $previous_datetime and then get the whole table
139   # starting from the second result to iterate through.
140   $previous_datetime = timestamp_to_the_nearest_whole_minute($prevdate);
141   $outputline = create_output_line( $previous_datetime, $current_mem_used );
```

```
142     verbose($outputline);
143     say OUTPUT_FILE $outputline if ($OUTPUT_TO_FILE);
144     while ( my $line = <INPUT_FILE> ) {
145       chomp($line);
146       next if $line =~ /^$/;
147       @columns = split( /,/, $line );
148       $current_mem_used = @columns[ $INPUT_HEADERS{memActuallyUsed} ] + @columns[ ↙
              ↳ $INPUT_HEADERS{swapActuallyUsed} ];
149       $current_datetime = @columns[ $INPUT_HEADERS{timestamp} ];
150       next if !check_if_in_given_range($current_datetime);
151 
152       $current_datetime = timestamp_to_the_nearest_whole_minute($current_datetime);
153 
154       # The following line ensures that we will avoid duplicate output lines
155       # due to timestamp_to_the_nearest_whole_minute. We might lose some data
156       # but this is rare and negligible in a huge table like the one we have
157       # meaning our final probabilities will not be affected.
158       next if ( $current_datetime == $previous_datetime );
159 
160       # Add missing lines every 1 minute.
161       my @missing_lines_time = get_missing_timestamps( $previous_datetime, $current_datetime, 60 );
162       if ( @missing_lines_time[0] != -1 ) {
163         foreach my $missing_timestamp (@missing_lines_time) {
164           $outputline = create_output_line($missing_timestamp);
165           verbose($outputline);
166           say OUTPUT_FILE $outputline if ($OUTPUT_TO_FILE);
167         }
168       }
169 
170       $outputline = create_output_line( $current_datetime, $current_mem_used );
171       verbose($outputline);
172       say OUTPUT_FILE $outputline if ($OUTPUT_TO_FILE);
173 
174       $previous_datetime = $current_datetime;
175     }
176 
177     close(OUTPUT_FILE) if ($OUTPUT_TO_FILE);
178 
179     my $shift_lines = abs( floor( $config{predict_future} / $config{data_collection_interval} ) );
180     $shift_lines = 1 if ( $shift_lines == 0 );
181     if ($OUTPUT_TO_FILE) {
182       my $command = $RShiftColumn_BIN . " '$OUTPUT_FILENAME' 'CMU' " . $shift_lines . " '" . ↙
              ↳ $bayesian_states{missing_values} . "'";
183       system($command);
184       #$command = $RShiftColumn_BIN . " '$OUTPUT_FILENAME' 'PMU' " . ( $shift_lines * 2 ) . " '" . ↙
              ↳ $bayesian_states{missing_values} . "'";
185       #system($command);
186     }
187 }
188 
189 # This function adds the missing data timestamps
190 # Inputs
191 # $previous_line <- a valid timestamp
192 # $current_line <- a valid timestamp
193 # $add_every_Xseconds <- Define the number of seconds in between
194 sub get_missing_timestamps {
195 
196     my ( $previous_line, $current_line, $add_every_Xseconds ) = @_;
197 
198     validate_timestamp($previous_line);
199     validate_timestamp($current_line);
200 
201     my @missing_lines;
202 
203     my $diff = timestamp_diff_in_seconds( $previous_line, $current_line );
204 
```

```
205     if ( $diff > $add_every_Xseconds ) {
206       for ( my $i = $add_every_Xseconds; $i < $diff; $i += $add_every_Xseconds ) {
207         push( @missing_lines, $previous_line += $add_every_Xseconds );
208       }
209       return @missing_lines;
210     }
211     return -1;
212
213 }
214
215 sub check_if_in_given_range {
216
217     my $timestamp = $_[0];
218
219     validate_timestamp($timestamp);
220
221     if ( $START_DATE != -1 and $END_DATE != -1 ) {
222       return 1 if ( $timestamp >= $START_DATE and $timestamp <= $END_DATE );
223     } elsif ( $START_DATE != -1 and $END_DATE == -1 ) {
224       return 1 if ( $timestamp >= $START_DATE );
225     } elsif ( $START_DATE == -1 and $END_DATE != -1 ) {
226       return 1 if ( $timestamp <= $END_DATE );
227     }
228     if ( $START_DATE == -1 and $END_DATE == -1 ) {
229       return 1;
230     } else {
231       return 0;
232     }
233
234 }
235
236 # http://www.linux-bsd-central.com/index.php/content/view/41/35/
237 # Remove temp files in case you received a kill signal
238 sub cleanAndExit {
239
240     say "\nCaught a kill signal, cleaning up temp files and exiting";
241
242     unlink($OUTPUT_FILENAME) if ($OUTPUT_TO_FILE);
243     exit 1;
244
245 }
246
247 sub usage {
248
249     say "Usage:";
250     say "-h Usage";
251     say "-v Verbose";
252     say "-d Debug";
253     say "-o Output Filename <- Create output file";
254     say "-S Start Date (can be a datetime or timestamp)";
255     say "-E End Date (can be a datetime or timestamp)";
256     say "-f Input file to read the data from.";
257
258 }
259
260 sub debug {
261
262     say $_[0] if $DEBUG;
263
264 }
265
266 sub verbose {
267
268     say $_[0] if $VERBOSE;
269
270 }
```

Listing A.8: insert-hosts-to-db.pl

```
#!/usr/bin/perl

# Vangelis Tasoulas <evanget@ifi.uio.no>

# Bayllocator -- Script to insert virtual machines to the database.

# Needed packages
use lib '/usr/local/lib/bayllocator';
use lib 'lib';
use lib '../lib';
use strict 'vars';
use Getopt::Std;
use feature 'say';
use Time::Local;
use POSIX qw(strftime);
use DBI;
use Sys::Virt;
use MyLibVirtSubs;
use MyConfig qw(%config);

# Global variables
my $VERBOSE = 0;
my $DEBUG = 0;

######################
# handle flags and arguments
# Example: c == "-c", c: == "-c argument"
my $opt_string = 'hvdas:';
getopts( "$opt_string", \my %opt ) or usage() and exit 1;

# print help message if -h is invoked
if ( $opt{'h'} ) {
   usage();
   exit 0;
}

$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};

my $dsn = "DBI:mysql:database=" . $config{db}{database} . ";host=" . $config{db}{host} . ";port=" . ↙
      ↳ $config{db}{port};
my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or die "Connection Error: ↙
      ↳ $DBI::errstr\n";

my $virtualMachineManager = Sys::Virt->new( address => $config{virsh_host} );

# Main program content

add_all_domains_in_db($virtualMachineManager);

# End opf Main program

# Disconnect from the database.
$db->disconnect();

######################
# Helper routines

# Return all active domains.

sub add_all_domains_in_db {

   my Sys::Virt($vmm) = shift(@_);
   my @all_domains = get_all_domains($vmm);

```

```
64    foreach my $dom (@all_domains) {
65
66      # If the entry does not exist, then add it to the database.
67      if ( entry_exists_in_db_hosts( $dom->get_uuid_string ) == 0 ) {
68        verbose( "Adding domain to db: " . $dom->get_name . ", uuid: " . $dom->get_uuid_string . ↙
              ↳ "\n" );
69
70        #<<< do not let perltidy touch this
71        my $sql = "INSERT INTO '" . $config{db}{hosts_table} . "'
72        ('name', 'uuid', 'active', 'date_host_added')
73        VALUES ('" . $dom->get_name . "', '" . $dom->get_uuid_string . "', 1, NOW());";
74        #>>>
75
76        $db->do($sql) or die "SQL Error: $DBI::errstr\n";
77      } else {
78        debug( "Entry exists in the database: " . $dom->get_name . ", uuid: " . $dom->get_uuid_string . ↙
              ↳ "\n" );
79      }
80
81    }
82
83  }
84
85  # Will return 0 if it doesn't exist or a positive number if it exists.
86  sub entry_exists_in_db_hosts {
87
88    my $uuid = $_[0];
89
90    my $sql = "SELECT * FROM '" . $config{db}{hosts_table} . "' WHERE 'uuid' = '$uuid';";
91    my $sth = $db->prepare($sql);
92    $sth->execute() or die "SQL Error: $DBI::errstr\n";
93    my $output = $sth->rows . "\n";
94    $sth->finish();
95
96    return $output;
97
98  }
99
100 sub usage {
101
102   # prints the correct use of this script
103   print "Usage:\n";
104   print "-h Usage\n";
105   print "-v Verbose\n";
106   print "-d Debug\n";
107   print "-a Add all available virtual hosts\n";
108   print "-s <domain name>";
109   print " Specify the name of the virtual host to add as listed by \"virsh list\" command\n";
110
111 }
112
113 sub verbose {
114
115   print $_[0] if ($VERBOSE);
116
117 }
118
119 sub debug {
120
121   print $_[0] if ($DEBUG);
122
123 }
```

Listing A.9: add-unix-sockets-to-vm-conf.pl

```
1  #!/usr/bin/perl
```

```
2
3  # Vangelis Tasoulas <evanget@ifi.uio.no>
4
5  # Script to add unix sockets for qemu-ga to the virtual machines
6
7  # Needed packages
8  use lib '/usr/local/lib/bayllocator';
9  use lib 'lib';
10 use lib '../lib';
11 use strict 'vars';
12 use Getopt::Std;
13 use feature 'say';
14 use Sys::Virt;
15 use MyLibVirtSubs;
16 use XML::Simple;
17 use Data::Compare;
18 use MyConfig qw(%config);
19
20 # Global variables
21 my $VERBOSE = 0;
22 my $DEBUG = 0;
23
24 ######################
25 # handle flags and arguments
26 # Example: c == "-c", c: == "-c argument"
27 my $opt_string = 'hvdg:';
28 getopts( "$opt_string", \my %opt ) or usage() and exit 1;
29
30 # print help message if -h is invoked
31 if ( $opt{'h'} ) {
32    usage();
33    exit 0;
34 }
35
36 $VERBOSE = 1 if $opt{'v'};
37 $DEBUG = 1 if $opt{'d'};
38
39 my $virtualMachineManager = Sys::Virt->new( address => $config{virsh_host} );
40
41 # Main program content
42
43 if ( $opt{'g'} ) {
44    my $domName = $opt{'g'};
45    my @inactive_domains = get_inactive_domain_names($virtualMachineManager);
46    my @all_domains = get_all_domain_names($virtualMachineManager);
47    my %inactive_domains_hash = map { $_ => 1 } @inactive_domains;
48    my %all_domains_hash = map { $_ => 1 } @all_domains;
49
50    if ( !exists( $all_domains_hash{$domName} ) ) {
51       say "$domName guest is not defined.";
52       exit 1;
53    } elsif ( !exists( $inactive_domains_hash{$domName} ) ) {
54       say "$domName guest is activated.";
55       say "Please shutdown the domain and try again.";
56       exit 1;
57    }
58 }
59
60 foreach my $dom ( get_inactive_domains($virtualMachineManager) ) {
61
62    #$memory_stats{'available'};
63    next if ( $opt{'g'} && $dom->get_name ne $opt{'g'} );
64    debug( "Adding Unix Sockets to guest: " . $dom->get_name );
65    my $myxmlcontroller = "<controller type='virtio-serial' index='0'>
66    <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
67    </controller>";
```

```
68    my $myxmlchannel1 = "<channel type='unix'>
69    <source mode='bind' path='/tmp/" . $dom->get_uuid_string . ".qemuga.sock'/>
70    <target type='virtio' name='org.qemu.guest_agent.0'/>
71    <address type='virtio-serial' controller='0' bus='0' port='1'/>
72    </channel>";
73    my $myxmlchannel2 = "<channel type='unix'>
74    <source mode='bind' path='/tmp/" . $dom->get_uuid_string . "-2.qemuga.sock'/>
75    <target type='virtio' name='org.qemu.guest_agent.1'/>
76    <address type='virtio-serial' controller='0' bus='0' port='2'/>
77    </channel>";
78
79    define_domain( $virtualMachineManager, add_device_to_domainxml( $dom->get_xml_description(), ↙
          ↳ $myxmlcontroller ) );
80    define_domain( $virtualMachineManager, add_device_to_domainxml( $dom->get_xml_description(), ↙
          ↳ $myxmlchannel1 ) );
81    define_domain( $virtualMachineManager, add_device_to_domainxml( $dom->get_xml_description(), ↙
          ↳ $myxmlchannel2 ) );
82  }
83
84  # End opf Main program
85
86  ####################
87  # Helper routines
88
89  # add_device_to_domainxml will add devices given in xml format
90  # to a domain's xml.
91  # The subroutine will return the xml text.
92  sub add_device_to_domainxml {
93    my $xmlDump = $_[0];
94    my $xmlAdd = $_[1];
95
96    my $xml = new XML::Simple;
97    my $xmlDumpData = $xml->XMLin( $xmlDump, KeepRoot => 1, ForceArray => 1 );
98
99    my $xmlAddData = $xml->XMLin( $xmlAdd, KeepRoot => 1, ForceArray => 1 );
100   foreach my $key ( keys %{$xmlAddData} ) {
101     if ( exists $$xmlAddData{$key} ) {
102       my $max_index = scalar @{ $$xmlAddData{$key} };
103
104       for ( my $i = 0; $i < $max_index; $i++ ) {
105
106         my $device_exists = 0;
107         foreach my $value ( values @{ $$xmlDumpData{domain}[0]{devices}[0]{$key} } ) {
108           if ( Compare( $value, $$xmlAddData{$key}[$i] ) ) {
109             $device_exists = 1;
110             break;
111           }
112         }
113         push( @{ $$xmlDumpData{domain}[0]{devices}[0]{$key} }, $$xmlAddData{$key}[$i] ) if ↙
               ↳ $device_exists == 0;
114       }
115     }
116   }
117
118   return $xml->XMLout( $xmlDumpData, KeepRoot => 1 );
119 }
120
121 sub usage {
122
123   # prints the correct use of this script
124   say "Usage:";
125   say "-h Usage";
126   say "-v Verbose";
127   say "-d Debug";
128   say "-g Choose Guest (Must be powered off)";
129   say " If '-g' option is not set, the script will try to update all of the inactive guests.";
```

```
}

sub verbose {
  say $_[0] if ($VERBOSE);
}

sub debug {
  say $_[0] if ($DEBUG);
}
```

Listing A.10: QueryNet.R

```
#!/usr/bin/Rscript

# Vangelis Tasoulas (evanget@ifi.uio.no)

# Bayesian Networks
# ./QueryNet.R '/etc/bayllocator/UbuntuServer1' MemUsedRate Monday h08 m00_04 rate_20_40 ↙
      ↳ mt_800_1000

args <- commandArgs(TRUE);

if (length(args) == 0) {
    cat("No command line arguments defined\n")
    q()
}

filename = args[1] # Needs the full path of the file.
node_to_get_answers = args[2]
evidence_weekday = args[3]
evidence_hour = args[4]
evidence_minute = args[5]
evidence_currentMemUsed = args[6]

if (! file.exists(filename)) {
    cat("Input file \"", filename, "\" does not exist!\n", sep="")
    q()
}

library(methods, quietly=TRUE, warn.conflicts = FALSE)
library(MASS, quietly=TRUE, warn.conflicts = FALSE)
library(RBGL, quietly=TRUE, warn.conflicts = FALSE)
library(igraph, quietly=TRUE, warn.conflicts = FALSE)
library(bnlearn, quietly=TRUE, warn.conflicts = FALSE)
library(gRbase, quietly=TRUE, warn.conflicts = FALSE)
library(gRain, quietly=TRUE, warn.conflicts = FALSE)

options(scipen=20)

mem.df <- read.table(filename, sep=",", header=TRUE);
mem.df[mem.df == "N/A"] = NA;

for (i in seq(1:length(mem.df))) {
  mem.df[[i]] <- factor(mem.df[[i]], exclude=NA)
}

### Define the Bayesian network here
mem <- empty.graph(nodes = names(mem.df));
arcs(mem) <- data.frame(
  from=c("W", "H", "M", "CMU"),
  to=c("FMU", "FMU", "FMU", "FMU")
);
###

assign("dag_from_bnlearn_network", NULL);
assign("myname", NULL);
```

```
assign("myparents", NULL);
for (i in seq(1,length(mem$nodes))) {
  myname <- names(mem$nodes)[[i]];
  myparents <- mem$nodes[[i]]$parents;
  if(length(myparents) != 0) {
    for(m in seq(1,length(myparents))) {
      dag_from_bnlearn_network <- rbind(dag_from_bnlearn_network, c(mem$nodes[[i]]$parents[[m]], myname))
    }
  }
}
remove(i, m, myname, myparents);

mem.dag <- ftM2graphNEL(ft=dag_from_bnlearn_network, edgemode="directed")

# Conditional probability table
mem.cpt <- extractCPT(mem.df, mem.dag, smooth = 0.0001)

# Create the network
mem.rgrainnet <- grain(compileCPT(mem.cpt))

prediction <- setFinding(
    mem.rgrainnet,
    nodes = c("W", "H", "M", "CMU"),
    states = c(evidence_weekday, evidence_hour, evidence_minute, evidence_currentMemUsed),
    propagate = FALSE
)

invisible(propagate(prediction))

answer <- querygrain(prediction, nodes = c(node_to_get_answers), type = "joint")

for (i in 1:length(answer)) {
    cat(paste(names(answer[i]), paste(answer[i], "\n", sep=""), sep=","))
}
```

Listing A.11: ShiftColumn.R

```
#!/usr/bin/Rscript

# Vangelis Tasoulas (evanget@ifi.uio.no)

# This script will read a csv file and it will
# shift one of the columns
# 1st Arg: CSV file to read (full path of the file)
# 2nd Arg: Number of column or column name to shift
# 3rd Arg: if a positive number, a positive shift will happen
# if a negative number, a negative shift will happen
# 4th Arg: Value to pass to the empty shifted values. It can be omitted.

# ./ShiftColumt.R '/etc/bayllocator/UbuntuServer1' MemUsedRate -5 "N/A"

args <- commandArgs(TRUE);

if (length(args) == 0) {
    cat("No command line arguments defined\n")
    q()
}

filename = args[1]
col = args[2]
numberoflines = args[3]
fillshifted = args[4]

if(length(fillshifted) == 0){fillshifted=NA}

```

```
if (! file.exists(filename)) {
    cat("Input file \"", filename, "\" does not exist!\n", sep="")
    q()
}

if (! is.numeric(as.numeric(numberoflines))) {
   cat("The NumberOfLinesToShift must be a numeric value")
   q()
}
numberoflines=as.numeric(numberoflines)

my.df <- read.table(filename, sep=",", header=TRUE);

if (! is.numeric(col)) {
   col<-match(col, colnames(my.df))
}

if(numberoflines > 0) {
   my.df[,col]<-c(rep(fillshifted,abs(numberoflines)), as.character(head(my.df[,col], -numberoflines)))
} else {
   my.df[,col]<-c(as.character(tail(my.df[,col], numberoflines)), rep(fillshifted,abs(numberoflines)))
}

write.table(my.df, file=filename, row.names = FALSE, sep=",", quote=FALSE )
```

Listing A.12: MyBayesianPredictor.pm

```
# A Package with the Bayesian related functions

# Vangelis Tasoulas <evanget@ifi.uio.no>

package MyBayesianPredictor;

use strict "vars";
use POSIX qw(strftime isdigit);
use Time::Local;
use feature 'say';
use MyConfig;
use MyTimeSubs;
use List::Util qw(sum);
use Data::Dumper;

require Exporter;
our @ISA = qw(Exporter);
our @EXPORT = qw(
   bayesian_predictor
   convertTimestampToBNStates
   convertMemToBNState
   convertRateToBNState
   create_output_line
   isnumeric
);

my $QUERYNET_BIN = "QueryNet.R";

sub bayesian_predictor {

   my $timestamp = $_[0];
   my $currentMemUsed = convertMemToBNState( $_[1], \%{ $bayesian_states{memory_util} } );
   my $domName = $_[2];
   my $full_file_path = $_[3];

   if ( !defined($full_file_path) ) {
      my $filename = $domName . $config{bayesian_state_files_ext};

      $full_file_path = $config{bayesian_state_files_path} . $filename;
```

```
40  }
41
42  if ( !-e $full_file_path ) {
43    say "No file found for guest '$domName'";
44    say "Expected file to be found: $full_file_path";
45    warn $!;
46    return -1;
47  }
48
49  my $timehash = convertTimestampToBNStates($timestamp);
50
51  my $day = $timehash->{Weekday};
52  my $hour = $timehash->{Hour};
53  my $minute = $timehash->{Minute};
54
55  # Call the R script to get a Bayesian prediction.
56  #say "$QUERYNET_BIN '$full_file_path' FMU $day $hour $minute $currentMemUsed";
57  my @probs_totalmem = `$QUERYNET_BIN '$full_file_path' FMU $day $hour $minute
        ↪ $currentMemUsed`;
58
59  my @TotalMem;
60  foreach my $prob (@probs_totalmem) {
61    chomp($prob);
62
63    # The $prob var looks like mt_900_1000,0.83
64    # This means that the system will use from 900-1000MB of memory
65    # with a probability of 83%
66    # Split $prob to comma.
67    my @results = split /,/, $prob;
68
69    my $min_key = 9999999999999999;
70    my $max_key = 0;
71    foreach my $key ( keys %{ $bayesian_states{memory_util} } ) {
72
73      # This loop will figure out the how much memory (an actual number)
74      # mem_800_1000 (or the given @results[0] variable) is.
75      # The $max_key will take the highest value of the key
76      # The $min_key will take the lowest value of the key
77      # For example:
78      # $bayesian_states{memory_total}{7} = mem_600_800
79      # $bayesian_states{memory_total}{8} = mem_800_1000
80      # $bayesian_states{memory_total}{9} = mem_800_1000
81      # $bayesian_states{memory_total}{10} = mem_1000_1200
82      # So the $min_key will take value 8 and the $max_key will take
83      # value 9. Then we add these numbers together plus 1,
84      # we multiply this to 100 and we get 900MB which is in the middle
85      # of 800 and 1000.
86      # (8 + 9 + 1) = 18
87      # 18 * 100 = 1800
88      # 1800 / 2 = 900
89      # Because we need the value in KB, we multiply what we found with 1024
90      if ( $bayesian_states{memory_util}{$key} eq @results[0] ) {
91        $min_key = $key if ( $key < $min_key );
92        $max_key = $key if ( $key > $max_key );
93      }
94    }
95
96    # Calculate the actual memory in KB
97    my $mem_in_kb = ( ( ( $min_key + $max_key + 1 ) * 100 ) / 2 ) * 1024;
98
99    # Multiply the actual memory with the probability of this event to occur and
100   # push this value in the @total_mem array.
101   push( @TotalMem, ( $mem_in_kb * @results[1] ) );
102 }
103
104 # Then the total memory is the summary of the @TotalMem array.
```

```
105     my $mem_expected = sprintf( "%.0f", sum(@TotalMem) );
106
107     return $mem_expected;
108 }
109
110 sub convertTimestampToBNStates {
111
112     my $timestamp = $_[0];
113
114     my %RetHash;
115
116     if ( isnumeric($timestamp) ) {
117         my $timehash = timestamp_to_discrete_hash($timestamp);
118
119         $RetHash{"Weekday"} = $bayesian_states{days}{ $$timehash{wday} };
120         $RetHash{"Hour"} = $bayesian_states{hours}{ $$timehash{hour} };
121         $RetHash{"Minute"} = $bayesian_states{minutes}{ $$timehash{min} };
122     } else {
123         $RetHash{"Weekday"} = $bayesian_states{missing_values};
124         $RetHash{"Hour"} = $bayesian_states{missing_values};
125         $RetHash{"Minute"} = $bayesian_states{missing_values};
126     }
127
128     return \%RetHash;
129
130 }
131
132 sub convertMemToBNState {
133
134     my $mem = $_[0];
135     my $mem_hash = $_[1];
136
137     my $rounded_mem = $mem;
138
139     # The $mem_avg_total is defined in KB, but we need to convert it to MB
140     # and round it to 100s of MB
141     if ( isnumeric($mem) ) {
142
143         $mem = $mem / 1024;
144
145         # $bayesian_states{memory_total} takes values from 2-60 this is why we divide by 100.
146         $mem = $mem / 100;
147
148         my $max_key = 0;
149         $_ > $max_key and $max_key = $_ for keys %{$mem_hash};
150
151         if ( $mem > $max_key ) {
152             $rounded_mem = sprintf( "%d", $max_key );
153         } else {
154             $rounded_mem = sprintf "%.0f", $mem;
155         }
156
157         return $$mem_hash{$rounded_mem};
158
159     } else {
160
161         return $bayesian_states{missing_values};
162
163     }
164
165 }
166
167 sub convertRateToBNState {
168
169     my $memrate = $_[0];
170
```

```
171     my $rounded_mem_rate = $memrate;
172
173     # If the data is defined and valid, add them to the $return_line
174     if ( isnumeric($memrate) ) {
175
176       my $max_key = 0;
177       $_ > $max_key and $max_key = $_ for keys %{ $bayesian_states{memory_util} };
178
179       if ( $memrate > $max_key ) {
180         $rounded_mem_rate = sprintf( "%.2f", $max_key );
181       } else {
182         $rounded_mem_rate = sprintf( "%.2f", $memrate );
183       }
184
185       return $bayesian_states{memory_util}{$rounded_mem_rate};
186
187     } else {
188
189       return $bayesian_states{missing_values};
190
191     }
192
193 }
194
195 sub create_output_line {
196
197     my ( $timestamp, $mem_used ) = @_;
198
199     # Convert the timestamp to discrete date values
200     my $timehash = convertTimestampToBNStates($timestamp);
201
202     $mem_used = convertMemToBNState( $mem_used, \%{ $bayesian_states{memory_util} } );
203
204     # Create the return results with the assumption that the collected data is in "N/A" state.
205     #my $return_line = join( ",", $timehash->{Weekday}, $timehash->{Hour}, $timehash->{Minute}, ↙
            ↳ $mem_used, $mem_used, $mem_used );
206     my $return_line = join( ",", $timehash->{Weekday}, $timehash->{Hour}, $timehash->{Minute}, ↙
            ↳ $mem_used, $mem_used );
207
208     #my $return_line = join( ",", $mem_used, $mem_used, $mem_used );
209
210     return $return_line;
211
212 }
213
214 # This function will check if the given variable is a valid
215 # float or integer with a potential leading plus or minus sign
216 # It will match
217 # 1235151
218 # +123234
219 # -421345
220 # .143234
221 # 12.345
222 sub isnumeric {
223
224     my $num = $_[0];
225
226     if ( $num =~ /^[+-]?(\.|\d+\.)?\d+$/ ) {
227       return 1;
228     } else {
229       return 0;
230     }
231 }
232
233 1;
```

Listing A.13: MyLibVirtSubs.pm

```
# A Package with custom LibVirt based subroutines
# used a lot in Bayllocator

# Vangelis Tasoulas <evanget@ifi.uio.no>

package MyLibVirtSubs;

use strict "vars";
use JSON;
use Sys::Virt;
use String::Random qw(random_regex random_string);
use IO::Socket::UNIX qw( SOCK_STREAM );
use Try::Tiny;
use Time::HiRes qw(sleep usleep nanosleep);
use MIME::Base64;
use IO::Select;

require Exporter;
our @ISA = qw(Exporter);
our @EXPORT = qw(
   get_active_domains
   get_inactive_domains
   get_all_domains
   get_active_domain_names
   get_inactive_domain_names
   get_all_domain_names
   define_domain
   qemuga_syncguest
   qemuga_readfile
);

# The following two lines are added only to get autocompletion durint the coding time.
# They can safely be commented out.
my %config = do('config.pl');
my $vmm = Sys::Virt->new( address => $config{virsh_host} );

sub get_active_domains {
   my Sys::Virt($vmm) = @_;
   my @active_domains = $vmm->list_domains();

   return @active_domains;
}

sub get_inactive_domains {
   my Sys::Virt($vmm) = @_;
   my @all_inactive_domains = $vmm->list_defined_domains();

   return @all_inactive_domains;
}

sub get_all_domains {
   my Sys::Virt($vmm) = @_;
   my @all_domains = get_active_domains($vmm);
   push( @all_domains, get_inactive_domains($vmm) );

   return @all_domains;
}

sub get_active_domain_names {
   my Sys::Virt($vmm) = @_;
   my @all_active_domains = get_active_domains($vmm);

   my @active_domain_names;
   foreach my $dom (@all_active_domains) {
      push( @active_domain_names, $dom->get_name );
```

```
66    }
67
68    return @active_domain_names;
69  }
70
71  sub get_inactive_domain_names {
72    my Sys::Virt($vmm) = @_;
73    my @all_inactive_domains = get_inactive_domains($vmm);
74
75    my @inactive_domain_names;
76    foreach my $dom (@all_inactive_domains) {
77      push( @inactive_domain_names, $dom->get_name );
78    }
79
80    return @inactive_domain_names;
81  }
82
83  sub get_all_domain_names {
84    my Sys::Virt($vmm) = @_;
85    my @all_domains = get_all_domains($vmm);
86
87    my @all_domain_names;
88    foreach my $dom (@all_domains) {
89      push( @all_domain_names, $dom->get_name );
90    }
91
92    return @all_domain_names;
93  }
94
95  sub define_domain {
96    my Sys::Virt($vmm) = $_[0];
97    my $xml = $_[1];
98
99    try {
100     $vmm->define_domain($xml);
101   }
102   catch {
103     warn "Cannot define domain: $_\n";
104     return -1;
105   };
106
107   return 0;
108 }
109
110 # qemu-ga Subs
111 # returns 0 if everything went alright, or -1 in any other case.
112 sub qemuga_syncguest {
113
114   # $sock is a UNIX socket
115   my ($sock) = @_;
116
117   if ( defined($sock) ) {
118
119     #print "qemuga_syncguest - Sock defined\n";
120     my $synced = 0;
121     my $globalCounter = 0;
122     while ( $synced == 0 ) {
123
124       #print "qemuga_syncguest - Into While, Synced: $synced\n";
125       my $guest_sync_str = random_regex('\d{18}');
126       my %JSON_QUERY = (
127         'execute' => 'guest-sync',
128         'arguments' => { 'id' => int $guest_sync_str }
129       );
130       my $query = encode_json( \%JSON_QUERY );
131
```

```
132         #print "qemuga_syncguest - My Query: $query\n";
133
134         if ( sock_send( $sock, $query ) != -1 ) {
135
136           #print "qemuga_syncguest - Query Sent to Socket\n";
137
138           my $counter = 0;
139           while ( $counter < 10 ) {
140
141             #print "qemuga_syncguest - Into While 2, Counter: $counter\n";
142             my $line = sock_recv_line($sock);
143
144             #print "qemuga_syncguest - Line: $line\n";
145             my $reply = try_decode_json($line);
146             if ( $reply != -1 && defined( $reply->{'return'} ) ) {
147               if ( $reply->{'return'} == $guest_sync_str ) {
148                 $synced = 1;
149                 last;
150               }
151             }
152             $counter++;
153           }
154         }
155         $globalCounter++;
156         return -1 if $globalCounter > 5;
157       }
158
159       return 0;
160     } else {
161       return -1;
162     }
163 }
164
165 # Read file and return its contents.
166 # Return -1 in any other case.
167 sub qemuga_readfile {
168   my ( $sock, $file ) = @_;
169
170   if ( defined $sock ) {
171     my %JSON_FILE_OPEN_QUERY = (
172       'execute' => 'guest-file-open',
173       'arguments' => {
174         'path' => $file,
175         'mode' => "r"
176       }
177     );
178
179     my $query = encode_json( \%JSON_FILE_OPEN_QUERY );
180     sock_send( $sock, $query );
181     my $line = sock_recv_line($sock);
182     my $reply = try_decode_json($line);
183     if ( $reply != -1 && defined( $reply->{"return"} ) ) {
184       my $handle = $reply->{"return"};
185
186       my %JSON_FILE_READ_QUERY = (
187         'execute' => 'guest-file-read',
188         'arguments' => {
189           'handle' => int $handle,
190           'count' => int 4096
191         }
192       );
193
194       $query = encode_json( \%JSON_FILE_READ_QUERY );
195       sock_send( $sock, $query );
196       $line = sock_recv_line($sock);
197       $reply = try_decode_json($line);
```

```
198         if ( $reply != -1 && defined( $reply->{'return'}{'buf-b64'} ) ) {
199           my $data = decode_base64( $reply->{'return'}{'buf-b64'} );
200
201           my %JSON_FILE_CLOSE_QUERY = (
202             'execute' => 'guest-file-close',
203             'arguments' => { 'handle' => int $handle, }
204           );
205
206           $query = encode_json( \%JSON_FILE_CLOSE_QUERY );
207           sock_send( $sock, $query );
208           sock_recv_line($sock);
209
210           return $data;
211         } else {
212           return -1;
213         }
214       } else {
215         return -1;
216       }
217     } else {
218       return -1;
219     }
220 }
221
222 #Returns the decoded json hash or -1 if failed to decode.
223 sub try_decode_json {
224     my $json_string = $_[0];
225
226     my $decoded_hash;
227
228     try {
229       $decoded_hash = decode_json($json_string);
230     }
231     catch {
232       warn "try_decode_json: $_\n";
233       return -1;
234     };
235
236     return $decoded_hash;
237 }
238
239 # Send a command to a socket
240 # Return 0 on success, -1 on failure.
241 sub sock_send {
242     my $sock = $_[0];
243     my $command = $_[1];
244
245     try {
246       print $sock $command . "\n";
247     }
248     catch {
249       warn "sock_send: $_\n";
250       return -1;
251     };
252
253     #print "sent\n";
254     return 0;
255 }
256
257 # returns the $line read.
258 sub sock_recv_line {
259     my $sock = $_[0];
260
261     my $line;
262
263     try {
```

```
264
265     # $line = <$sock>;
266     # chomp $line;
267     my $sel = IO::Select->new($sock);
268     while ( my @has_buf = $sel->can_read(0.1) ) {
269       foreach my $has_buf_i (@has_buf) {
270
271         ## Create another select to can read byte by byte, because
272         ## we just know that the socket have data, but not the length:
273         my $sel = IO::Select->new($has_buf_i);
274
275         ## Read byte by byte:
276         while ( $sel->can_read(0.01) ) {
277           sysread( $has_buf_i, $line, 1, length($line) );
278         }
279       }
280     }
281   }
282   catch {
283     warn "sock_send: $_\n";
284     return;
285   };
286   return $line;
287 }
288
289 1;
```

Listing A.14: MyTimeSubs.pm

```
1  # A Package with many datetime and timestamp
2  # manipulation subroutines used a lot in
3  # Bayllocator
4
5  # Vangelis Tasoulas <evanget@ifi.uio.no>
6
7  package MyTimeSubs;
8
9  use strict "vars";
10 use POSIX qw(strftime isdigit);
11 use Time::Local;
12 use feature 'say';
13
14 require Exporter;
15 our @ISA = qw(Exporter);
16 our @EXPORT = qw(
17   datetime_to_timestamp
18   timestamp_to_datetime
19   add_minutes_to_timestamp
20   add_minutes_to_datetime
21   datetime_to_the_nearest_whole_minute
22   timestamp_to_the_nearest_whole_minute
23   datetime_diff_in_seconds
24   timestamp_diff_in_seconds
25   datetime_to_discrete_hash
26   timestamp_to_discrete_hash
27   datetime_to_YmdHMS
28   validate_datetime
29   validate_timestamp
30 );
31
32 sub datetime_to_timestamp {
33
34   my $datetime = $_[0];
35
36   my ( $year, $month, $day, $hours, $minutes, $seconds ) = datetime_to_YmdHMS($datetime);
37
```

```
38    return ( timelocal( $seconds, $minutes, $hours, $day, ( $month - 1 ), $year ) );
39
40  }
41
42  sub timestamp_to_datetime {
43
44    my $timestamp = $_[0];
45
46    return strftime( '%Y-%m-%d %H:%M:%S', localtime($timestamp) );
47
48  }
49
50  sub add_minutes_to_datetime {
51
52    my $datetime = $_[0];
53    my $minadd = $_[1];
54
55    return timestamp_to_datetime( add_minutes_to_timestamp( datetime_to_timestamp($datetime), ↙
          ↳ $minadd ) );
56
57  }
58
59  sub add_minutes_to_timestamp {
60
61    my $timestamp = $_[0];
62    my $minutes = $_[1];
63
64    return $timestamp + ( $minutes * 60 );
65
66  }
67
68  # This function will return the given datetime to the nearest whole minute.
69  # If 2012-03-19 23:32:08 then 2012-03-19 23:32:00 will be returned
70  # If 2012-03-19 23:32:38 then 2012-03-19 23:33:00 will be returned
71  sub datetime_to_the_nearest_whole_minute {
72
73    my $datetime = $_[0];
74
75    return timestamp_to_datetime( timestamp_to_the_nearest_whole_minute( ↙
          ↳ datetime_to_timestamp($datetime) ) );
76
77  }
78
79  # This function will return a timestamp of the given timestamp to the nearest whole minute.
80  sub timestamp_to_the_nearest_whole_minute {
81
82    my $timestamp = $_[0];
83
84    my $seconds = $timestamp % 60;
85
86    if ( $seconds >= 30 ) {
87      $timestamp += ( 60 - $seconds );
88    } else {
89      $timestamp -= $seconds;
90    }
91
92    return $timestamp;
93
94  }
95
96  sub datetime_diff_in_seconds {
97
98    my $time1 = $_[0];
99    my $time2 = $_[1];
100
101   return timestamp_diff_in_seconds( datetime_to_timestamp($time1), datetime_to_timestamp($time2) );
```

```
102
103 }
104
105 sub timestamp_diff_in_seconds {
106
107     my $time1 = $_[0];
108     my $time2 = $_[1];
109
110     return abs( $time1 - $time2 );
111
112 }
113
114 # datetime_to_discrete_hash() will return a hash with the discrete date values
115 # from timestamp_to_discrete_hash()
116 sub datetime_to_discrete_hash {
117
118     my $datetime = $_[0];
119
120     return timestamp_to_discrete_hash( datetime_to_timestamp($datetime) );
121
122 }
123
124 # timestamp_to_discrete_hash() will return a hash with the following discrete date values
125 # sec: 0-59
126 # min: 0-59
127 # hour: 0-23
128 # mday: 1-31
129 # mon: 1-12
130 # year: YYYY
131 # wday: 0-6, 0 = Sunday
132 # yday: 1-366
133 # isdst: True if the specified time occurs during Daylight Saving Time, false otherwise.
134 sub timestamp_to_discrete_hash {
135
136     my $timestamp = $_[0];
137
138     my %DATA;
139
140     ( $DATA{sec}, $DATA{min}, $DATA{hour}, $DATA{mday}, $DATA{mon}, $DATA{year}, ↙
        ↳ $DATA{wday}, $DATA{yday}, $DATA{isdst} ) = localtime($timestamp);
141
142     $DATA{mon} += 1;
143     $DATA{year} += 1900;
144     $DATA{yday} += 1;
145
146     return \%DATA;
147
148 }
149
150 # If the datetime is not valid die
151 sub datetime_to_YmdHMS {
152
153     my $datetime = $_[0];
154
155     if ( $datetime =~ /^(\d{4})-(\d\d)-(\d\d)\s+(\d\d):(\d\d):(\d\d)$/ ) {
156
157         my $year = $1;
158         my $month = $2;
159         my $day = $3;
160         my $hours = $4;
161         my $minutes = $5;
162         my $seconds = $6;
163
164         return ( $year, $month, $day, $hours, $minutes, $seconds );
165
166     } else {
```

```
167
168     die "It is not a datetime. Should be 'YYYY/mm/dd HH:MM:SS'";
169
170   }
171
172 }
173
174 # Datetime validation. Return 1 if datetime is valid, or dies
175 # in any other case (basically an alias of datetime_to_YmdHMS())
176 sub validate_datetime {
177
178   datetime_to_YmdHMS( $_[0] );
179
180   return 1;
181
182 }
183
184 sub validate_timestamp {
185
186   die "It is not a timestamp" if !isdigit( $_[0] );
187
188 }
189
190 1;
```

Listing A.15: MyDBSubs.pm

```
1  # A Package with many datetime and timestamp
2  # manipulation subroutines used a lot in
3  # Bayllocator
4
5  # Vangelis Tasoulas <evanget@ifi.uio.no>
6
7  package MyDBSubs;
8
9  use strict "vars";
10 use POSIX qw(strftime isdigit);
11 use Time::Local;
12 use feature 'say';
13 use MyTimeSubs;
14 use MyConfig qw(%config);
15
16 require Exporter;
17 our @ISA = qw(Exporter);
18 our @EXPORT = qw(
19   get_host_ids_from_db
20   get_hostname_by_id
21   get_hostname_by_Uuid
22   get_hostid_by_name
23   get_hostid_by_Uuid
24   get_hostUuid_by_id
25   get_hostUuid_by_name
26   get_max_rows_of_table
27   get_max_rows_of_table_per_host
28   get_row_number_of_table_by_id
29 );
30
31 my $dsn = "DBI:mysql:database=" . $config{db}{database} . ";host=" . $config{db}{host} . ";port=" . ↙
       ↳ $config{db}{port};
32
33 # Returns an array with ids from all of the
34 # available hosts in hosts_table
35 sub get_host_ids_from_db {
36
37   my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or warn "Connection Error: ↙
         ↳ $DBI::errstr\n";
```

```
38
39    my @ids;
40
41    #<<< do not let perltidy touch this
42    my $sql = "SELECT id
43    FROM $config{db}{hosts_table}";
44    #>>>
45    my $sth = $db->prepare($sql);
46    $sth->execute() or die "SQL Error: $DBI::errstr\n";
47    while ( my $ref = $sth->fetchrow_hashref() ) {
48      push( @ids, $ref->{'id'} );
49    }
50    $sth->finish();
51
52    $db->disconnect();
53
54    return @ids;
55
56  }
57
58  sub get_hostname_by_id {
59
60    my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or warn "Connection Error: ↙
          ↳ $DBI::errstr\n";
61
62    my $hostId = $_[0];
63
64    if ( defined($db) ) {
65
66      #<<< do not let perltidy touch this
67      my $name = $db->selectrow_array("SELECT name
68                                       FROM $config{db}{hosts_table}
69                                       WHERE id = '$hostId'
70                                       LIMIT 1");
71      #>>>
72      $db->disconnect();
73
74      return $name;
75
76    } else {
77
78      warn "No database defined";
79      return -1;
80
81    }
82
83  }
84
85  sub get_hostname_by_Uuid {
86
87    my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or warn "Connection Error: ↙
          ↳ $DBI::errstr\n";
88
89    my $hostUuid = $_[0];
90
91    if ( defined($db) ) {
92
93      #<<< do not let perltidy touch this
94      my $name = $db->selectrow_array("SELECT name
95                                       FROM $config{db}{hosts_table}
96                                       WHERE uuid = '$hostUuid'
97                                       LIMIT 1");
98      #>>>
99      $db->disconnect();
100
101     return $name;
```

```
102
103     } else {
104
105         warn "No database defined";
106         return -1;
107
108     }
109
110 }
111
112 sub get_hostid_by_name {
113
114     my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or warn "Connection Error: ↙
            ↳ $DBI::errstr\n";
115
116     my $hostName = $_[0];
117
118     if ( defined($db) ) {
119
120         #<<< do not let perltidy touch this
121         my $id = $db->selectrow_array("SELECT id
122                                        FROM $config{db}{hosts_table}
123                                        WHERE name = '$hostName'
124                                        LIMIT 1");
125         #>>>
126         $db->disconnect();
127
128         return $id;
129
130     } else {
131
132         warn "No database defined";
133         return -1;
134
135     }
136
137 }
138
139 sub get_hostid_by_Uuid {
140
141     my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or warn "Connection Error: ↙
            ↳ $DBI::errstr\n";
142
143     my $hostUuid = $_[0];
144
145     if ( defined($db) ) {
146
147         #<<< do not let perltidy touch this
148         my $id = $db->selectrow_array("SELECT id
149                                        FROM $config{db}{hosts_table}
150                                        WHERE uuid = '$hostUuid'
151                                        LIMIT 1");
152         #>>>
153         $db->disconnect();
154
155         return $id;
156
157     } else {
158
159         warn "No database defined";
160         return -1;
161
162     }
163
164 }
165
```

```
166 sub get_hostUuid_by_id {
167
168     my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or warn "Connection Error: ↙
          ↳ $DBI::errstr\n";
169
170     my $hostId = $_[0];
171
172     if ( defined($db) ) {
173
174         #<<< do not let perltidy touch this
175         my $uuid = $db->selectrow_array("SELECT uuid
176                                          FROM $config{db}{hosts_table}
177                                          WHERE id = '$hostId'
178                                          LIMIT 1");
179         #>>>
180         $db->disconnect();
181
182         return $uuid;
183
184     } else {
185
186         warn "No database defined";
187         return -1;
188
189     }
190
191 }
192
193 sub get_hostUuid_by_name {
194
195     my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or warn "Connection Error: ↙
          ↳ $DBI::errstr\n";
196
197     my $hostName = $_[0];
198
199     if ( defined($db) ) {
200
201         #<<< do not let perltidy touch this
202         my $uuid = $db->selectrow_array("SELECT uuid
203                                          FROM $config{db}{hosts_table}
204                                          WHERE name = '$hostName'
205                                          LIMIT 1");
206         #>>>
207         $db->disconnect();
208
209         return $uuid;
210
211     } else {
212
213         warn "No database defined";
214         return -1;
215
216     }
217
218 }
219
220 sub get_max_rows_of_table {
221
222     my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or warn "Connection Error: ↙
          ↳ $DBI::errstr\n";
223
224     my $table = $_[0];
225
226     if ( defined($db) ) {
227
228         # Make a quick selection from $config{db}{data_table} to determine the total number of rows.
```

```
229     #<<< do not let perltidy touch this
230     my $sql = "SELECT 'id'
231     FROM '$table';";
232     #>>>
233     my $sth = $db->prepare($sql);
234     $sth->execute();
235 
236     # Store this value to the $max_rows var.
237     my $max_rows = $sth->rows();
238     $sth->finish();
239 
240     $db->disconnect();
241 
242     return $max_rows;
243 
244   } else {
245 
246     warn "No database defined";
247     return -1;
248 
249   }
250 
251 }
252 
253 # Return the total number of rows per host
254 # from the average table
255 sub get_max_rows_of_table_per_host {
256 
257   my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or warn "Connection Error: ↙
          ↳ $DBI::errstr\n";
258 
259   my $table = $_[0];
260   my $hostid = $_[1];
261   my $query_add = $_[2];
262 
263   if ( defined($db) ) {
264 
265     # Make a quick selection from $config{db}{data_table} to determine the total number of rows.
266     #<<< do not let perltidy touch this
267     my $sql = "SELECT id
268     FROM $table
269     WHERE data_host_id = $hostid $query_add;";
270     #>>>
271     my $sth = $db->prepare($sql);
272     $sth->execute();
273 
274     # Store this value to the $max_rows var.
275     my $max_rows = $sth->rows();
276     $sth->finish();
277 
278     $db->disconnect();
279 
280     return $max_rows;
281 
282   } else {
283 
284     warn "No database defined";
285     return -1;
286 
287   }
288 
289 }
290 
291 # Return the total number of rows the data table contains until the given id number
292 sub get_row_number_of_table_by_id {
293 
```

```
294     my $db = DBI->connect( $dsn, $config{db}{user}, $config{db}{pw} ) or warn "Connection Error: ↙
            ↳ $DBI::errstr\n";
295
296     my $table = $_[0];
297     my $id = $_[1];
298
299     if ( defined($db) ) {
300
301         #<<< do not let perltidy touch this
302         my $sql = "SELECT 'id'
303         FROM '$table'
304         WHERE 'id' <= $id;";
305         #>>>
306         my $sth = $db->prepare($sql);
307         $sth->execute();
308
309         # Store this value to the $max_rows var.
310         my $id_actual_pos = $sth->rows();
311         $sth->finish();
312
313         $db->disconnect();
314
315         return $id_actual_pos;
316
317     } else {
318
319         warn "No database defined";
320         return -1;
321
322     }
323
324 }
325
326 1;
```

Listing A.16: MyEMailNotifier.pm

```
1  # A Package to send template based notification
2  # e-mails for Bayllocator using the mail system
3  # command from mailutils or bsd-mailx
4
5  # Vangelis Tasoulas <evanget@ifi.uio.no>
6
7  package MyEMailNotifier;
8
9  use strict "vars";
10 use POSIX qw(strftime isdigit);
11 use Time::Local;
12 use feature 'say';
13 use MyConfig;
14 use MyTimeSubs;
15
16 require Exporter;
17 our @ISA = qw(Exporter);
18 our @EXPORT = qw(
19     send_email_to_admins
20     send_email
21 );
22
23 # You can define more directories using a semicolon
24 my $bayllocator_confdir = "/etc/bayllocator;../conf";
25 my $email_suffix = "-email.template";
26
27 # This function will send an e-mail to all of the defined admins in the
28 # configuration file.
29 sub send_email_to_admins {
```

```
30
31    my $message = $_[0];
32    my $email_type = $_[1];
33
34    foreach my $recipient ( @{ $config{admin}{email} } ) {
35       send_email( $recipient, $message, $email_type );
36    }
37
38 }
39
40 sub send_email {
41
42    my $recipient = $_[0];
43
44    # $message contains the message to be e-maild
45    my $message = $_[1];
46
47    # $email_type contains a keyword to match the template to be used.
48    # You can create your own templates under /etc/bayllocator/*
49    # And their name must follow the convention: "$email_type . $email_suffix".
50    my $email_type = $_[2];
51
52    my $subject = "Bayllocator Email System";
53
54    my $template = get_template($email_type);
55
56    if ( $template == -1 ) {
57
58       $message = "No valid template found in any of the defined directories.";
59       $subject = "Bayllocator: No valid template found";
60       call_mail( $recipient, $message, $subject );
61
62    } else {
63
64       my $final_message;
65
66       open( TEMPLATE, $template ) or warn("Could not open template: $template\n$!") and return;
67       while ( my $line = <TEMPLATE> ) {
68          if ( $line =~ /^<Subject:(.*)>/ ) {
69             $subject = $1;
70             next;
71          }
72          $final_message .= $line if ( $line !~ /^\s*#/ );
73       }
74       close TEMPLATE;
75
76       my $timestamp = timestamp_to_datetime(time);
77
78       $final_message =~ s/<first_name>/$config{admin}{first_name}/g;
79       $final_message =~ s/<last_name>/$config{admin}{last_name}/g;
80       $final_message =~ s/<email_message>/$message/g;
81       $final_message =~ s/<timestamp>/$timestamp/g;
82
83       $subject =~ s/<first_name>/$config{admin}{first_name}/g;
84       $subject =~ s/<last_name>/$config{admin}{last_name}/g;
85       $subject =~ s/<email_message>/$message/g;
86       $subject =~ s/<timestamp>/$timestamp/g;
87
88       call_mail( $recipient, $final_message, $subject );
89    }
90
91 }
92
93 # Find if template exists in paths.
94 # Return the template full path or -1 if not found.
95 sub get_template {
```

```
96
97    my $email_type = $_[0];
98
99    my @dirs = split /;/, $bayllocator_confdir;
100
101   my $template;
102
103   for ( my $i = 0; $i < ( scalar @dirs ); $i++ ) {
104     if ( -d @dirs[$i] ) {
105       $template = @dirs[$i] . "/" . $email_type . $email_suffix;
106       last if ( -f $template );
107     } else {
108       $template = -1;
109     }
110   }
111
112   return $template;
113
114 }
115
116 # Issuing the mail system command to send the e-mail
117 sub call_mail {
118
119   my $recipient = $_[0];
120   my $message = $_[1];
121   my $subject = $_[2];
122
123   system("echo \'$message\'|mail -s \'$subject\' $recipient");
124
125   #system("echo \"$message\"|mail -s \"$subject\" $recipient");
126
127 }
128
129 1;
```

Listing A.17: MyConfig.pm

```
1  # A Package with to send template based
2  # notification e-mails for Bayllocator
3
4  # Vangelis Tasoulas <evanget@ifi.uio.no>
5
6  package MyConfig;
7
8  use lib '/usr/local/lib/bayllocator';
9  use lib '/etc/bayllocator';
10 use lib 'conf';
11 use lib 'lib';
12 use lib '../conf';
13 use lib '../lib';
14 use lib '.';
15
16 require 'bayllocator.cfg';
17 our ( %config, %bayesian_states );
18
19 require Exporter;
20 our @ISA = qw(Exporter);
21 our @EXPORT = qw(
22   %config
23   %bayesian_states
24 );
25
26 1;
```

Listing A.18: bayllocator.cfg

```
#!/usr/bin/perl

# Vangelis Tasoulas <evanget@ifi.uio.no>

# Bayllocator Configuration

#-----------------------------------------------------

%config;
%bayesian_states;

# --- Admin info ---

# Set the name and the e-mail of the person whom you
# would like to receive e-mail notifications
$config{admin} = {
   first_name => 'Vangelis',
   last_name => 'Tasoulas',
   email => [ 'vangelis@tasoulas.net', 'evanget@ifi.uio.no' ]
};

#-----------------------------------------------------

# --- Database ---

# Change the configuration in this section to match the
# settings of your database.
$config{db} = {
   host => 'prox1.iu.hio.no',
   port => '33306',
   database => 'bayesian_data',
   hosts_table => 'hosts',
   data_table => 'data',
   rolling_avg_table => 'averages',
   changes_from_last_sampling_table => 'changes_view',
   user => 'username',
   pw => 'password'
};

#-----------------------------------------------------

# --- Data Collection and Prediction ---

# Define the virsh host.
# So far it only works for the local host, so do not change this variable.
$config{virsh_host} = 'qemu:///system';

# Define how much memory ( given in KB ) should
# be guaranteed to the hypervisor. The rest will
# be available for the virtual machines!
$config{hypervisor_memory} = ( 20 * 1024 * 1024 ); # (2 * 1024 * 1024) = 2048 MB

# Define the minimum memory the guests should have.
$config{minimum_guest_memory} = ( 300 * 1024 );

# Define the interval given in seconds, of sample
# collection from virtual machines.
$config{data_collection_interval} = 60;

# Path to save the Bayesian State files including
# following slash "/"
# DO NOT FORGET THE "/" FOLLOWING
# Make also sure that you have write access to the
# defined directory.
$config{bayesian_state_files_path} = '/etc/bayllocator/';
```

```
66
67  # The extension to use for the bayesian state files
68  # stored in the above location.
69  # If a guest with the name "MyGuest" exists,
70  # Then the expected file to be found is :
71  # "$config{bayesian_state_files}/MyGuest$config{bayesian_state_files_ext}"
72  $config{bayesian_state_files_ext} = '.dat';
73
74  # Define the interval which bayllocator should be
75  # running ( given in seconds ). A value of 300 means
76  # that bayllocator should try to reallocate memory
77  # to the guests every 5 minutes.
78  # ###### Important ######
79  # For more accurate predictions, please whenever you
80  # change this variable, consider changing the
81  # $config{averages_min} variable as well and make
82  # your database is up to date.
83  $config{action_interval} = 300;
84
85  # Define the number of minutes for rolling averages.
86  # Please make sure that the averages table into the
87  # database has the appropriate columns:
88  # mem_avg_used_$config{averages_min}
89  # mem_avg_total_$config{averages_min}
90  # mem_used_rate_$config{averages_min}
91  $config{averages_min} = 5;
92
93  # Defines how far in the future should Bayllocator predict.
94  # By default this value is equals to $config{action_interval}
95  $config{predict_future} = $config{action_interval};
96
97  # Define how much more than the predicted memory bayllocator
98  # should allocate to the virtual machines.
99  # A value of 0.15 means add 15%. If the prediction "says" 100MB
100 # the guest will need 100MB, then give it 115MB (100 + 100 * 15%)
101 $config{add_to_prediction} = 0.15;
102
103 #--------------------------------------------------------
104
105 # --- Bayesian State Match ---
106
107 # The variables in this section, must match the state
108 # names of the Bayesian network nodes
109 # You might need to add more similar hashes here if
110 # your Bayesian network expands, but then you might
111 # need to make small changes to the file
112 # data-to-bn-states.pl
113 #$bayesian_states{headers} = ["Weekday", "Hour", "Minute", "MemCurrentlyUsed", ↙
        ↳ "MemPreviouslyUsed", "MemToUse"];
114 $bayesian_states{headers} = [ "W", "H", "M", "CMU", "FMU" ];
115
116 $bayesian_states{missing_values} = 'N/A';
117
118 $bayesian_states{days} = {
119    1 => '1', # Monday
120    2 => '1', # Tuesday
121    3 => '1', # Wednesday
122    4 => '1', # Thursday
123    5 => '1', # Friday
124    6 => '0', # Saturday
125    0 => '0' # Sunday
126 };
127
128 %{ $bayesian_states{hours} } = map {
129    $a = sprintf( "h%02d", $_ );
130    $_ => $a;
```

```
} 0 .. 23;

%{ $bayesian_states{minutes} } = map {
   $a = 'm00_04' if /0/ .. /4/;
   $a = 'm05_09' if /5/ .. /9/;
   $a = 'm10_14' if /10/ .. /14/;
   $a = 'm15_19' if /15/ .. /19/;
   $a = 'm20_24' if /20/ .. /24/;
   $a = 'm25_29' if /25/ .. /29/;
   $a = 'm30_34' if /30/ .. /34/;
   $a = 'm35_39' if /35/ .. /49/;
   $a = 'm40_44' if /40/ .. /44/;
   $a = 'm45_49' if /45/ .. /49/;
   $a = 'm50_54' if /50/ .. /54/;
   $a = 'm55_59' if /55/ .. /59/;
   $_ => $a;
} 0 .. 59;

%{ $bayesian_states{memory_util} } = map {
   $temp = $_ * 100;
   $a = 'mem_100_less' if $temp < 100;
   $a = 'mem_4000_greater' if $temp >= 4000;
   for ( my $i = 100; $i <= 3900; $i += 100 ) {
      if ( $temp >= $i and $temp < ( $i + 100 ) ) {
         $a = sprintf( "mem_%d_%d", $i, ( $i + 100 ) );
         break;
      }
   }
   $_ => $a;
} 0 .. 40;

#--------------------------------------------------------

# Do not remove this 1;
1;
```

Listing A.19: warn-email.template

```
######################################################################
# Lines starting with a hash are omitted
#
# Define the Subject in the first line like this:
# <Subject:Bayllocator Warning> : Bayllocator Warning is the
# Subject here
#
# Available parameters in templates:
# <first_name> : Will be replaced with the first name
# defined in bayllocator.cfg
# <last_name> : Will be replaced with the last name
# defined in bayllocator.cfg
# <email_message> : Will be replaced with the message to
# be e-mailed
# <timestamp> : Will be replaced with the current date
# and time the e-mail created
######################################################################
####
<Subject:Bayllocator Warning>
####
Hello <first_name>,

This e-mail was sent at <timestamp>.

<email_message>

Yours truly,
Your bayllocator server
```

Listing A.20: fatal-email.template

```
#################################################################
# Lines starting with a hash are omitted
#
# Define the Subject in the first line like this:
# <Subject:Bayllocator Warning> : Bayllocator Warning is the
# Subject here
#
# Available parameters in templates:
# <first_name> : Will be replaced with the first name
# defined in bayllocator.cfg
# <last_name> : Will be replaced with the last name
# defined in bayllocator.cfg
# <email_message> : Will be replaced with the message to
# be e-mailed
# <timestamp> : Will be replaced with the current date
# and time the e-mail created
#################################################################
####
<Subject:Bayllocator: ## Important ## Something might have died!>
####
Hello <first_name>,

We have an urgent situation!!
Something might have crashed...

This e-mail was sent at <timestamp>.
More information follows:

<email_message>

Yours truly,
Your bayllocator server
```

Listing A.21: bayllocator.sql

```
-- MySQL dump 10.13 Distrib 5.1.61, for debian-linux-gnu (x86_64)
--
-- Host: localhost Database: bayllocator
-- ------------------------------------------------------
-- Server version 5.1.61-0ubuntu0.11.10.1

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
    ↳ FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table 'averages'
--

DROP TABLE IF EXISTS 'averages';
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE 'averages' (
```

```
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `data_id` int(11) NOT NULL,
  `data_host_id` smallint(6) NOT NULL,
  `date` timestamp NULL DEFAULT NULL,
  `mem_avg_used_5` double(10,3) DEFAULT NULL COMMENT 'Average Used Memory of the following 5 minutes.\n\nNOTE: This represents actual memory plus actual swap in use.',
  `mem_avg_total_5` double(10,3) DEFAULT NULL COMMENT 'Average Total Memory of the following 5 minutes.\nSince we will use ballooning, the total memory will be changing a lot.\n\nNOTE: This is only the available total RAM',
  `mem_used_rate_5` double(10,3) DEFAULT NULL COMMENT 'A rate of how much memory is used for the following 5 minutes',
  PRIMARY KEY (`id`,`data_id`,`data_host_id`),
  UNIQUE KEY `data_id_UNIQUE` (`data_id`),
  KEY `date` (`date`),
  KEY `fk_averages_data1` (`data_id`,`data_host_id`),
  CONSTRAINT `fk_averages_data1` FOREIGN KEY (`data_id`, `data_host_id`) REFERENCES `data` (`id`, `host_id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=66861 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `averages`
--

LOCK TABLES `averages` WRITE;
/*!40000 ALTER TABLE `averages` DISABLE KEYS */;
/*!40000 ALTER TABLE `averages` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `data`
--

DROP TABLE IF EXISTS `data`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `data` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `host_id` smallint(6) NOT NULL,
  `date` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  `mem_total` double(10,3) DEFAULT NULL,
  `mem_used` double(10,3) DEFAULT NULL,
  `swap_total` double(10,3) DEFAULT NULL,
  `swap_used` double(10,3) DEFAULT NULL,
  `CPU_count` smallint(6) DEFAULT NULL,
  `load_avg_1` float DEFAULT NULL,
  `load_avg_5` float DEFAULT NULL,
  `load_avg_15` float DEFAULT NULL,
  `uptime` double(10,3) DEFAULT NULL,
  `cpuIdlingTime` double(10,3) DEFAULT NULL,
  `rxBytesTotal` bigint(20) DEFAULT NULL,
  `txBytesTotal` bigint(20) DEFAULT NULL,
  `rxPacketsTotal` bigint(20) DEFAULT NULL,
  `txPacketsTotal` bigint(20) DEFAULT NULL,
  PRIMARY KEY (`id`,`host_id`),
  KEY `fk_data_hosts` (`host_id`),
  KEY `date_memtot_hosts` (`date`,`mem_total`,`host_id`),
  CONSTRAINT `fk_data_hosts` FOREIGN KEY (`host_id`) REFERENCES `hosts` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=145930 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `data`
--

```

```
LOCK TABLES `data` WRITE;
/*!40000 ALTER TABLE `data` DISABLE KEYS */;
/*!40000 ALTER TABLE `data` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `hosts`
--

DROP TABLE IF EXISTS `hosts`;
/*!40101 SET @saved_cs_client = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `hosts` (
  `id` smallint(6) NOT NULL AUTO_INCREMENT,
  `name` varchar(45) DEFAULT NULL,
  `uuid` varchar(45) DEFAULT NULL,
  `active` tinyint(1) DEFAULT NULL,
  `date_host_added` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`),
  UNIQUE KEY `uuid_UNIQUE` (`uuid`)
) ENGINE=InnoDB AUTO_INCREMENT=13 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `hosts`
--

LOCK TABLES `hosts` WRITE;
/*!40000 ALTER TABLE `hosts` DISABLE KEYS */;
/*!40000 ALTER TABLE `hosts` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2012-03-23 1:57:36
```

Listing A.22: install-uninstall

```
#!/bin/bash
#set -x
# Vangelis Tasoulas <evanget@ifi.uio.no>

# Bayllocator -- Installation Script

if [ "`whoami`" != "root" ]; then

   echo "You need to become root to run this installation"
   exit 1

fi

BIN_TARGET="/usr/local/bin"
LIB_TARGET="/usr/local/lib/bayllocator"
CFG_TARGET="/etc/bayllocator"

#-------------------------------------------------------------

BIN_SRC="bin"
LIB_SRC="lib"
```

```
22 CFG_SRC="conf"
23
24 # Arrays of the files to be installed
25 BINS_TO_INSTALL=( add-unix-sockets-to-vm-conf.pl bayllocator.pl collect-data.pl ⤶
       ↳ data-to-bn-states.pl data-smoothing.pl loadsim.pl insert-hosts-to-db.pl ⤶
       ↳ check-missing-data.pl predictor.pl QueryNet.R ShiftColumn.R ⤶
       ↳ data-from-file-to-bn-states.pl )
26
27 LIBS_TO_INSTALL=( MyLibVirtSubs.pm MyTimeSubs.pm MyEMailNotifier.pm MyConfig.pm ⤶
       ↳ MyDBSubs.pm MyBayesianPredictor.pm )
28
29 CFGS_TO_INSTALL=( bayllocator.cfg fatal-email.template warn-email.template )
30
31 ACTION=$(basename $0)
32
33 # Copy files from a source folder to a destination
34 # Inputs:
35 # $1: SRC_DIR. The dir to read the files from
36 # $2: DST_DIR. The destination directory for the file
37 # $3: VERBOSE. If user should be notified of files being
38 # copied, set VERBOSE to 1. If not, set it to 0.
39 # $@: A file list with the files to be copied from the
40 # source dir to the destination dir
41 function copy_files_in_dir {
42
43   local SRC_DIR=$1
44   shift
45   local DST_DIR=$1
46   shift
47   local VERBOSE=$1
48   shift
49   local FILELIST=( $@ )
50
51   if [ ! -d "$SRC_DIR" ]; then
52     return 255
53   fi
54
55   if [ ! -d "$DST_DIR" ]; then
56     return 255
57   fi
58
59   local MAX=${#FILELIST[*]}
60   local MAX_LOOP=$(expr ${MAX} - 1)
61   pushd "$SRC_DIR" > /dev/null
62   for i in $(seq 0 $MAX_LOOP); do
63     if [ -f "${FILELIST[i]}" ]; then
64       if [ $VERBOSE -eq 1 ]; then
65         echo "Copying ${FILELIST[i]} to $DST_DIR"
66       fi
67       cp ${FILELIST[i]} $DST_DIR
68     fi
69   done
70   popd > /dev/null
71
72 }
73
74 # Remove files from a destination folder
75 # Inputs:
76 # $1: SRC_DIR. The dir to read the files from
77 # $2: DST_DIR. The destination directory for the file
78 # $3: VERBOSE. If user should be notified of files being
79 # copied, set VERBOSE to 1. If not, set it to 0.
80 # $@: A file list with the files to be copied from the
81 # source dir to the destination dir
82 function remove_files_in_dir {
83
```

```
84    local DST_DIR=$1
85    shift
86    local VERBOSE=$1
87    shift
88    local FILELIST=( $@ )
89
90    if [ ! -d "$DST_DIR" ]; then
91      return 255
92    fi
93
94    local MAX=${#FILELIST[*]}
95    local MAX_LOOP=$(expr ${MAX} - 1)
96    pushd "$DST_DIR" > /dev/null
97    for i in $(seq 0 $MAX_LOOP); do
98      if [ -f "${FILELIST[i]}" ]; then
99        if [ $VERBOSE -eq 1 ]; then
100         echo "Removing ${FILELIST[i]} from $DST_DIR"
101       fi
102       rm -f ${FILELIST[i]}
103     fi
104   done
105   popd > /dev/null
106
107   # If DST_DIR is empty, remove it.
108   if [ ! "$(ls -A $DST_DIR)" ]; then
109     if [ $VERBOSE -eq 1 ]; then
110       echo "$DST_DIR is empty. Removing.."
111     fi
112     rm -rf "$DST_DIR"
113   else
114     if [ $VERBOSE -eq 1 ]; then
115       echo ""
116       echo "$DST_DIR is not empty."
117       echo "Remove the contents of the folder and remove it manually if you want to"
118       echo ""
119     fi
120   fi
121
122 }
123
124 # Check if files exist in folder
125 # Inputs:
126 # $1: Dir to check if the files exist
127 # $2: Boolean (0 or 1) to define if there will be verbose output
128 # Set this to 1 if you want to let the user know of the missing
129 # files
130 # $@: The file list
131 # Returns:
132 # 255 if directory does not exist
133 # 0 if all of them exist
134 # 1 if even one (or more but none all) of them does not exist
135 # 2 if none of them exist
136 function check_files_in_dir {
137
138   local MY_DIR=$1
139   shift
140   local VERBOSE=$1
141   shift
142   local MY_FILELIST=( $@ )
143
144   if [ ! -d "$MY_DIR" ]; then
145     return 255
146   fi
147
148   local MAX=${#MY_FILELIST[*]}
149   local MAX_LOOP=$(expr ${MAX} - 1)
```

```
150     local RETURN=0
151     local COUNT_NON_EXISTING=0
152     pushd "$MY_DIR" > /dev/null
153     for i in $(seq 0 $MAX_LOOP); do
154       if [ ! -f "${MY_FILELIST[i]}" ]; then
155         if [ $VERBOSE -eq 1 ]; then
156           echo "$(pwd)/${MY_FILELIST[i]} not found"
157         fi
158         COUNT_NON_EXISTING=$(expr $COUNT_NON_EXISTING + 1)
159       fi
160     done
161     popd > /dev/null
162
163     if [ $COUNT_NON_EXISTING -eq $MAX ]; then
164       RETURN=2
165     elif [ $COUNT_NON_EXISTING -gt 0 ]; then
166       RETURN=1
167     fi
168
169     return $RETURN
170   }
171
172   function check_setup {
173
174     check_files_in_dir $BIN_SRC 1 ${BINS_TO_INSTALL[@]}
175     if [ $? -ge 1 ]; then echo "Check that your installation files are in place"; exit 1; fi
176     check_files_in_dir $LIB_SRC 1 ${LIBS_TO_INSTALL[@]}
177     if [ $? -ge 1 ]; then echo "Check that your installation files are in place"; exit 1; fi
178     check_files_in_dir $CFG_SRC 1 ${CFGS_TO_INSTALL[@]}
179     if [ $? -ge 1 ]; then echo "Check that your installation files are in place"; exit 1; fi
180
181   }
182
183   # Function to check if the software is already installed.
184   # If it is installed (or leftovers), it will return 1 otherwise 0
185   function check_install {
186
187     local RETURN=1
188
189     check_files_in_dir $BIN_TARGET 0 ${BINS_TO_INSTALL[@]}
190     if [ $? -ge 2 ]; then RETURN=0; fi
191     check_files_in_dir $LIB_TARGET 0 ${LIBS_TO_INSTALL[@]}
192     if [ $? -ge 2 ]; then RETURN=0; fi
193
194     return $RETURN
195
196   }
197
198   # Function to check if the configuration already exists.
199   # If it exists, will return 1 otherwise 0
200   function check_conf {
201
202     local RETURN=1
203
204     check_files_in_dir $CFG_TARGET 0 ${CFGS_TO_INSTALL[@]}
205     if [ $? -eq 2 ]; then RETURN=0; fi
206
207     return $RETURN
208
209   }
210
211   # Evaluate user input between the given $ACCEPTED_ARGS
212   # and return 1 if it valid or 0 if invalid.
213   function evaluate_user_input {
214
215     local USER_INPUT=$1
```

```
216   shift
217   local ACCEPTED_ARGS=( $@ )
218   local VALID=0
219
220   for IN in ${ACCEPTED_ARGS[@]}; do
221     if [ "$USER_INPUT" == "$IN" ]; then
222       VALID=1
223       break
224     fi
225   done
226
227   return $VALID
228
229 }
230
231 # Returns:
232 # 0 if No
233 # 1 if Yes
234 # 2 if Cancel
235 # 3 if None of them
236 function evaluate_yes_no_cancel {
237
238   local INPUT=$1
239   local YES="y Y yes Yes"
240   local NO="n N no No"
241   local CANCEL="Cancel C c"
242
243   for IN in ${NO}; do
244     if [ "$INPUT" == "$IN" ]; then
245       return 0
246     fi
247   done
248
249   for IN in ${YES}; do
250     if [ "$INPUT" == "$IN" ]; then
251       return 1
252     fi
253   done
254
255   for IN in ${CANCEL}; do
256     if [ "$INPUT" == "$IN" ]; then
257       return 2
258     fi
259   done
260
261   return 3
262 }
263
264 # This function will ask for user input and it will
265 # only accept yes or no for it. It will evaluate
266 # user's input against evaluate_yes_no_cancel function
267 # and return what the evaluate_yes_no_cancel function returns
268 function ask_for_yes_no {
269
270   local VALID_INPUT=0
271   local ACCEPTED_INPUTS="y n Y N yes no Yes No"
272
273   while [ $VALID_INPUT -eq 0 ]; do
274     read INPUT
275     while [ "$INPUT" == "" ]; do
276       read INPUT
277     done
278     evaluate_user_input $INPUT $ACCEPTED_INPUTS
279     VALID_INPUT=$?
280   done
281
```

```
282   evaluate_yes_no_cancel $INPUT
283   local RETURN=$?
284   echo ""
285
286   return $RETURN
287
288 }
289
290 function install_bayllocator {
291
292   check_setup
293
294   echo ""
295   echo "You are about to install Bayllocator"
296   echo "The default paths selected are:"
297   echo " Binaries: $BIN_TARGET"
298   echo " Libraries: $LIB_TARGET"
299   echo " Configuration: $CFG_TARGET"
300   echo ""
301   echo "Do you want to continue? (yes/no)"
302
303   ask_for_yes_no
304   if [ $? -eq 1 ]; then
305
306     check_install
307     if [ $? -eq 1 ]; then
308
309       echo ""
310       echo "Looks like a previous installation already exists."
311       echo "This script will remove it to be able to continue."
312       echo "Previous configuration will be kept intact. You will"
313       echo "be asked later if you want to overwrite it during the"
314       echo "installation. If not, you might need to update it with"
315       echo "the latest version bundled with this installation."
316       echo ""
317       echo "Do you want to go on? (yes/no)"
318
319       ask_for_yes_no
320       if [ $? -eq 1 ]; then
321         uninstall_bayllocator force
322       else
323         echo "Installation aborted"
324         exit 1
325       fi
326
327     fi
328
329     echo "Creating folder $BIN_TARGET"
330     mkdir -p $BIN_TARGET
331     echo "Creating folder $LIB_TARGET"
332     mkdir -p $LIB_TARGET
333     echo "Creating folder $CFG_TARGET"
334     mkdir -p $CFG_TARGET
335     copy_files_in_dir "$BIN_SRC" "$BIN_TARGET" 1 ${BINS_TO_INSTALL[@]}
336     copy_files_in_dir "$LIB_SRC" "$LIB_TARGET" 1 ${LIBS_TO_INSTALL[@]}
337
338     check_conf
339     if [ $? -eq 1 ]; then
340       echo ""
341       echo "Existing configuration files found in $CFG_TARGET."
342       echo ""
343       echo "Do you want to overwrite them? (yes/no)"
344       ask_for_yes_no
345       if [ $? -eq 1 ]; then
346         copy_files_in_dir "$CFG_SRC" "$CFG_TARGET" 1 ${CFGS_TO_INSTALL[@]}
347       fi
```

```
348       else
349         copy_files_in_dir "$CFG_SRC" "$CFG_TARGET" 1 ${CFGS_TO_INSTALL[@]}
350       fi
351
352       echo "installation finished"
353     else
354       echo "Installation aborted by user."
355       exit 1
356     fi
357
358 }
359
360 # This function will uninstall bayllocator
361 # It will give a notice to the user first
362 # except if "force" is passed as an argument
363 function uninstall_bayllocator {
364
365     local FORCE=0
366     local UNINSTALL=0
367     local KEEPCONF=1
368
369     check_install
370     if [ $? -eq 1 ]; then
371       if [ "$1" == "force" ]; then FORCE=1; fi
372
373       if [ $FORCE -eq 0 ]; then
374         echo ""
375         echo "You are about to remove Bayllocator."
376         echo ""
377         echo "Are you sure? (yes/no)"
378
379         ask_for_yes_no
380         if [ $? -eq 1 ]; then
381           UNINSTALL=1
382           echo "Do you want to remove your configuration files as well? (yes/no)"
383           ask_for_yes_no
384           if [ $? -eq 1 ]; then
385             KEEPCONF=0
386           fi
387         fi
388       elif [ $FORCE -eq 1 ]; then
389         UNINSTALL=1
390         KEEPCONF=1
391       fi
392     else
393       echo "Bayllocator is not installed"
394       exit 1
395     fi
396
397     if [ $UNINSTALL -eq 1 ]; then
398       if [ $FORCE -eq 1 ]; then
399         remove_files_in_dir "$BIN_TARGET" 0 ${BINS_TO_INSTALL[@]}
400         remove_files_in_dir "$LIB_TARGET" 0 ${LIBS_TO_INSTALL[@]}
401       else
402         remove_files_in_dir "$BIN_TARGET" 1 ${BINS_TO_INSTALL[@]}
403         remove_files_in_dir "$LIB_TARGET" 1 ${LIBS_TO_INSTALL[@]}
404         if [ $KEEPCONF -eq 0 ]; then
405           remove_files_in_dir "$CFG_TARGET" 1 ${CFGS_TO_INSTALL[@]}
406         fi
407       fi
408     else
409       echo "Uninstallation aborted by user"
410       exit 1
411     fi
412
413 }
```

```
414
415 ################################
416 ################################
417 ################################
418 # Main program content #
419 ################################
420 ################################
421 ################################
422
423 if [ "$ACTION" == "install" ]; then
424
425     install_bayllocator
426
427 elif [ "$ACTION" == "uninstall" ]; then
428
429     uninstall_bayllocator
430
431 else
432
433     echo ""
434     echo "Cannot recognise the action to be performed"
435     echo "Needs to be either 'install' or 'remove'"
436     echo ""
437
438 fi
439
440 exit 0
```